VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra kybernetiky a biomedicínského inženýrství

Implementace RFID technologie v nízkoodběrových vestavěných zařízeních

Implementation of RFID technology in low power embedded systems

2016 Vojtěch Baránek

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra kybernetiky a biomedicínského inženýrství

Zadání bakalářské práce

Student: **Vojtěch Baránek**

Studijní program: B2649 Elektrotechnika

Studijní obor: 2612R041 Řídicí a informační systémy

Téma: Implementace RFID technologie v nízkoodběrových vestavěných zařízeních
Implementation of RFID Technology in Low Power Embedded Systems

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Popis RFID technologie.
2. Popis technologie Freescale Kinetis L.
3. Rešerše algoritmů pro nízkoodběrové systémy.
4. Návrh systému s RFID technologií.
5. Realizace řešení pomocí vývojových kitů.
6. Testování řešení a zhodnocení výsledků práce.

Seznam doporučené odborné literatury:

[1] PRAUZEK, M., P. MUSILEK, A. G. WATTS a M. MICHALIKOVA. Powering environmental monitoring systems in arctic regions: A simulation study. *Elektronika ir Elektrotechnika.* 2014, 20 (7), pp. 34-37.
[2] WATTS, A. G., M. PRAUZEK, P. MUSILEK, E. PELIKAN a A. SANCHEZ-AZOFEIFA. Fuzzy power management for environmental monitoring systems in tropical regions. In: *Proceedings of the International Joint Conference on Neural Networks.* . Piscataway : IEEE Computer Society, 2014, s. 1719-1726. ISBN 978-1-4799-1484-5. DOI: 10.1109/IJCNN.2014.6889844.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Michal Prauzek, Ph.D.**

Datum zadání: 01.09.2015
Datum odevzdání: 29.04.2016

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA Fakulta elektrotechniky a informatiky OSTRAVA

doc. Ing. Jiří Koziorek, Ph.D.
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 29. dubna 2016 .....

Poděkování

Chtěl bych poděkovat panu Ing. Michalovi Prauzkovi, Ph.D. za poskytnutí skvělého tématu bakalářské práce a za užitečné konzultace. Taktéž chci poděkovat panu Ing. Jaromíru Konečnému, Ph.D. za pomoc ve věcech vývoje SW. A konečně chci poděkovat panu Ing. Martinu Stankušovi za technickou podporu.

Abstrakt

Tato bakalářská práce se zaměřuje na vývoj nízkoodběrového měřicího zařízení pro dlouhodobá měření využívajícího technologii RFID, potažmo NFC. V rámci kontextu popisuje v teoretické části samotnou tuto technologii. Taktéž popisuje technologii Freescale Kinetis L kvůli zvolenému mikrokontroléru. Dále popisuje algoritmy pro dosažení nízkoodběrovosti, které jsou později využity. V praktické části je proveden návrh nového systému s využitím dříve popsaných teoretických znalostí a vzápětí postup jeho fyzické realizace. Na konci jsou rekapitulovány původní cíle a je zhodnoceno jejich dosažení, jakožto i praktická využitelnost a přínos celé práce.

Klíčová slova: RFID/NFC, nízkoodběrovost, měřicí systém, Freescale Kinetis L, STMicroelectronics, Android

Abstract

This bachelor thesis focuses on developing a low power measuring device for long-time measurements using the technology of RFID, NFC respectively. According to the context it describes the chosen technology in its theoretical part. It also describes Freescale Kinetis L technology because of the selected microcontroller unit. Last but not least it describes algorithms for reaching the low power feature, which are used later. In the practical part a new system is designed using the previously described theoretical knowledge, and then it is implemented. At the very end the original goals are recapitulated and their achievement is assessed as well as the practical usability and its merit.

Key Words: RFID/NFC, low power, measuring system, Freescale Kinetis L, STMicroelectronics, Android

Obsah

Seznam použitých zkratek a symbolů

A/D	–	Analog/Digital
CMOS	–	Complementary metal-oxide-semiconductor
CSV	–	Comma-separated values
DPS	–	Deska plošných spojů
GPIO	–	General Purpose Input/Output
I^2C	–	Inter-Integrated Circuit
LCD	–	Liquid-crystal display
MCU	–	Microcontroller unit
MIME	–	Multipurpose Internet Mail Extensions
MOSFET	–	Metal-oxide-semiconductor field-effect transistor
NDEF	–	NFC Data Exchange Format
NFC	–	Near field communication
OTG	–	On-The-Go
OS	–	Operační systém
PE	–	Processor Expert
RFID	–	Radio-frequency identification
SPI	–	Serial Peripheral Bus
UART	–	Universal asynchronous receiver/transmitter
URI	–	Uniform Resource Identifier
USB	–	Universal Serial Bus

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Tato práce se zabývá analýzou, návrhem a realizací nízkoodběrového měřicího zařízení s využitím technologie RFID, potažmo NFC. NFC je aktuálně se rozvíjející technologie, která má velký potenciál. Již dnes se zvyšuje její přítomnost v reálném světě a bezpochyby bude mít v budoucnu nemalý význam a stane se součástí mnoha technických aplikací. Téma této práce představuje příležitost do této problematiky nahlédnout a využít ji nápaditým způsobem.

Práce začíná teoretickou částí, kde nejprve popisuje RFID technologii jako takovou a v návaznosti technologii NFC. V další části pojednává o sérii nízkoodběrových mikrokontrolérů Freescale Kinetis L. Poté pokračuje analýzou algoritmů pro dosažení nízkoodběrovosti. Na teoretickou část navazuje část praktická, kde je provedena analýza a návrh konkrétního systému a vzápětí samotná realizace se zhodnocením výsledků. Snahou je dosáhnout hmatatelného úspěchu, který bude mít praktický přínos.

2 RFID technologie

2.1 Původ názvu

Zkratka RFID pochází z angličtiny (Radio-frequency identification), v překladu znamená identifikaci na rádiové frekvenci. Slovo identifikace samo naznačuje hlavní využití této technologie, a to identifikaci věcí díky zjištění jejich přiděleného identifikátoru a porovnáním s určitou databází. Zmíněnými věcmi mohou být výrobky, zboží, movitý majetek osob či přímo osoby samotné.

2.2 Historické pozadí

První patent na tuto technologii byl udělen Charlesi Waltonovi roku 1983. K velkému rozvoji této technologie zásadně přispěla společnost Wal-Mart Stores provozující řetězec obchodních domů, která má přirozenou potřebu sofistikovaně chránit své zboží před opuštěním obchodního domu bez příslušné platby.

2.3 Rozdělení prvků technologie

Technologie je standardizovaná a popisuje ji mezinárodní norma ISO/IEC 18000. Ta jako základní stavební jednotky definuje čtečky (reader) a štítky (tag). Je příhodnější používat počeštěnou variantu slova tag, nežli slovo štítek.

2.3.1 Tagy

Tag je nositelem informace, má v závislosti na typu vlastní non-volatilní paměť, ve které jsou uložena užitečná data malého objemu (jedná se typicky o řád desítek bajtů), nejčastěji je to pouhý jedinečný identifikátor, který má význam až po konfrontaci s přiřazenými daty v databázi.

Tagy se dají rozdělit do skupin podle několika hledisek. Z pohledu zápisu do paměti, se dělí na přepisovatelné (R/W), zapsatelné (PROM) a nepřepisovatelné (ROM):

- Přepisovatelné mají často paměť EEPROM a jejich data se dají volně měnit.
- Zapsatelné jsou z výroby prázdné a dovolují jedenkrát zapsat data, která už pak nelze měnit.
- Nepřepisovatelné jsou již vyrobeny již se zapsanými daty, paměť má tedy permanentní charakter a nelze nijak měnit.

Z hlediska napájení se dělí na aktivní, aktivní s podporou baterie (BAP) a pasivní:

- Aktivní tagy periodicky vysílají svou informaci do okolí, bez ohledu na to, jestli ji někdo přijímá. Tomuto principu se v angličtině říká *beacon*, v překladu maják.

- Aktivní s podporou baterie jsou trvale ve stand-by režimu, dokud nedetekují vnější elektromagnetické pole rádiové komunikace, v tu chvíli začnou svou informaci aktivně vysílat tak dlouho, dokud pole stále detekují.
- Pasivní nemají žádný vlastní zdroj energie a energii vnějšího elektromagnetické pole z rádiového spojení. Díky tomu nepotřebují žádnou údržbu jako je kontrola elektrochemického zdroje a mají tedy v podstatě neomezenou životnost. Samozřejmě na druhou stranu kladou požadavky na minimální intenzitu pole, což v praxi znamená, že jsou použitelné na kratší vzdálenosti maximálně v řádu jednotek metrů.

2.3.2 Čtečky

Čtečky představují uzlové body systému, neboť je jejich úkolem získávat informace od tagů a nějakým způsobem je zpracovávat, často jsou totiž čtečky napojeny na počítač nebo určitou síť.

Dají se rozdělit na pasivní a aktivní:

- Pasivní čtečky nevysílají do okolí, protože ani nemají takovou schopnost, jen čekají na příjem signálu. Díky tomu jim stačí menší anténa nižšího výkonu, což znamená i menší rozměr zařízení.
- Aktivní čtečky nepřetržitě vysílají signál do svého okolí a snaží se zpět přijímat odpověď. Potřebují silnější anténu s vyšším výkonem.

2.4 Systémové rozdělení

Předchozí způsoby rozdělení nejsou nejvýstižnější samy o sobě, je lepší nahlížet na systém jako celek a zavést rozdělení podle jeho oboustranné činnosti. Jedna ze stran bývá pohyblivá a druhá statická. Systémy se dají dělit na systém *aktivní čtečka pasivní tag*, *pasivní čtečka aktivní tag* a *aktivní čtečka aktivní tag*:

- Systém *pasivní čtečka aktivní tag* je takový, kde aktivní tag pracuje v režimu majáku a neustále vysílá svou informaci a pasivní čtečka nepřetržitě čeká na příjem signálu. Pohyblivá zde může být kterákoliv ze stran. Díky pasivitě může mít čtečka snížené rozměry.
- V systému *aktivní čtečka aktivní tag* se používá buďto přímo aktivní tag nebo lépe pasivní tag s podporou baterie. Čtečka má pak za úkol tento tag probudit, což vyvolá z jeho strany vysílání své informace, kterou vzápětí čtečka přijme.
- Systém *aktivní čtečka pasivní tag* klade požadavek na anténu čtečky, která musí být dostatečně výkonná, aby pokryla požadovanou oblast signálem s dostatečnou intenzitou tak, aby stačila k napájení tagu. Tag však na druhou stranu může být malých rozměrů, zato nemůže v případě možnosti rychlého pohybu nést objemnější informaci, protože by hrozilo, že ji nestihne celou odvysílat.

2.5 NFC

Zkratka NFC znamená v angličtině *near field communication*, v překladu *komunikace v blízkém poli*. Myšlenka stojící za vznikem NFC je taková, že po vzoru RFID by mohla přenosná zařízení pomocí rádiového spojení zprostředkovaného zabudovanou anténou komunikovat s jinými přenosnými i nepřenosnými zařízeními, a to na krátkou vzdálenost a i s možností zabezpečení. Vzniklo více projektů takového zaměření, ale všechny selhaly, až právě na NFC.

Velmi obezřetně společně firmy NXP Semiconductors, Sony a Nokia založily neziskovou asociaci NFC Forum, jejíž úkolem je sdružovat firmy podílející se na NFC technologii a tuto technologii normalizovat zaváděním standardů, aby nemuselo NFC projít fází masivní nekompatibility. Dnes NFC Forum sdružuje přibližně 200 firem.

NFC je tedy standard popsaný normami ISO/IEC 14443, ISO/IEC 18092 a pak těmi vydanými NFC Forum. Navazuje na RFID, ovšem jeho vymezení je užší. Využívá smyčkových antén na frekvenci 13,56 MHz, komunikace je úmyslně limitovaná dosahem maximálně na 10 cm, v praxi bývá dosah poloviční nebo ještě menší.

Podobně jako u RFID se entity dají rozdělit na čtečky a tagy. Je zde však jednodušší dělení, neboť čtečky jsou vždy aktivní a tagy pasivní. Pro čtečky není určeno další rozdělení. Základní rozdělení tagů je popsáno standardem ISO/IEC 14443, který je dělí podle komunikačního rozhraní na typ A a B, které se liší ve způsobu modulace a kódování přenosu. Zároveň existují další standardy, a to MIFARE, Calypso a FeliCa. Ucelenější rozdělení zavádí standard NFC Forum, který rozlišuje 4 typy:

- Typ 1 stojí na typu A standardu ISO/IEC 14443. Paměť bývá do velikosti 1 kB, přenosová rychlost je 106 kbit s^{-1}. Zabezpečení je možné heslem o 32 nebo 64 bitech. Užívá se pro jednoúčelové tagy.
- Typ 2 se velmi podobá typu 1, ale může mít až dvojnásobnou velikost paměti. Naopak nepodporuje žádné zabezpečení. Využitím jsou opět jednoúčelové tagy.
- Typ 3 je poměrně speciální, stojí na standardu FeliCa. Paměť může mít velikost až 1 MB, přenosová rychlost je 212 kbit s^{-1}. Zabezpečení je možné heslem o 32 nebo 64 bitech. Na tento typ se specializuje společnost Sony, tagy mají široký způsob využití a patří k těm nejdražším.
- Typ 4 je postaven na obou typech standardu ISO/IEC 14443, konkrétní série tagů vždy vyhovuje buď typu A nebo B. Paměť je limitovaná na 64 kB, přenosová rychlost činí 424 kbit s^{-1}. Zabezpečení je volitelné, heslo nebývá delší než 128 bitů. Tagy jsou obecného účelu a jsou cenově dostupnější, než typ 3.

Pro výměnu dat mezi NFC entitami definuje NFC Forum formát výměny dat NDEF *(NFC Data Exchange Format)*. Základní jednotkou je NDEF záznam, který obsahuje typ obsažených dat,

jejich délku a samotná data. Typ může být některý z definovaných NFC Forum, URI, MIME nebo i neznámý. Záznamy stejného druhu se slučují do NDEF zpráv. Celá problematika je dosti složitá a cílí na optimalizaci komunikace.

Z celkového pohledu se zavádí systémové dělení, které souvisí s tím, jaký účel má NFC v dané situaci a tedy v jakém režimu pracuje:

- Režim reader/writer je takový, že čtečka přistupuje k datům tagu. Komunikaci upravuje standard ISO/IEC 14443 nebo FeliCa. Čtečka samozřejmě poskytuje tagu zdroj energie, jedná se tedy o nejméně úsporný režim.
- Režim peer-to-peer je takový, že komunikují 2 čtečky a vyměňují si libovolná data. Komunikaci upravuje standard NFCIP-1, který zavádí základní mechanismy v linkové vrstvě. Existuje LLCP protokol, který NFCIP-1 značně rozšiřuje chování linkové vrstvy o pokročilé mechanismy.
- Režim card emulation je specialitou NFC, neboť se čtečka vydává za tag a komunikace probíhá jako při čtecí operaci v režimu reader/writer.

3 Freescale Kinetis L

Nejprve je nutno zmínit se o výrobci. V názvu je firma Freescale, která je autorem řady Kinetis L, dnes již však tato firma neexistuje, neboť došlo k fúzi s firmou NXP, kdy nová společnost převzala název NXP a název Freescale byl opuštěn. Díky tomu je dnes firma NXP gigantem a jedním z nejvýznamnějších hráčů na poli mikrokontrolérů (MCU) s uplatněním od jednoúčelových aplikací, přes síťové prvky až po dnešní mobilní telefony. V této práci bude však stále používán název firmy Freescale, jakožto tehdejšího výrobce.

3.1 Charakteristika

Freescale Kinetis L je řada objemná řada mikrokontrolérů s několika variantními větvemi, která je postavená na procesoru ARM Cortex-M0+, kdy se jedná o 32-bitovou architekturu. Zároveň tato řada s využitím vlastností použitého procesoru akcentuje na velmi nízkou spotřebu při dostatečném výkonu, čímž se zaměřuje na takové aplikace, kde je jako zdroj energie užit akumulátor, ale přitom má tato aplikace požadavek na výpočetní výkon. Jak bylo zmíněno, kritickou oporou je samotný procesor ARM Cortex-M0+.

3.1.1 ARM Cortex-M0+

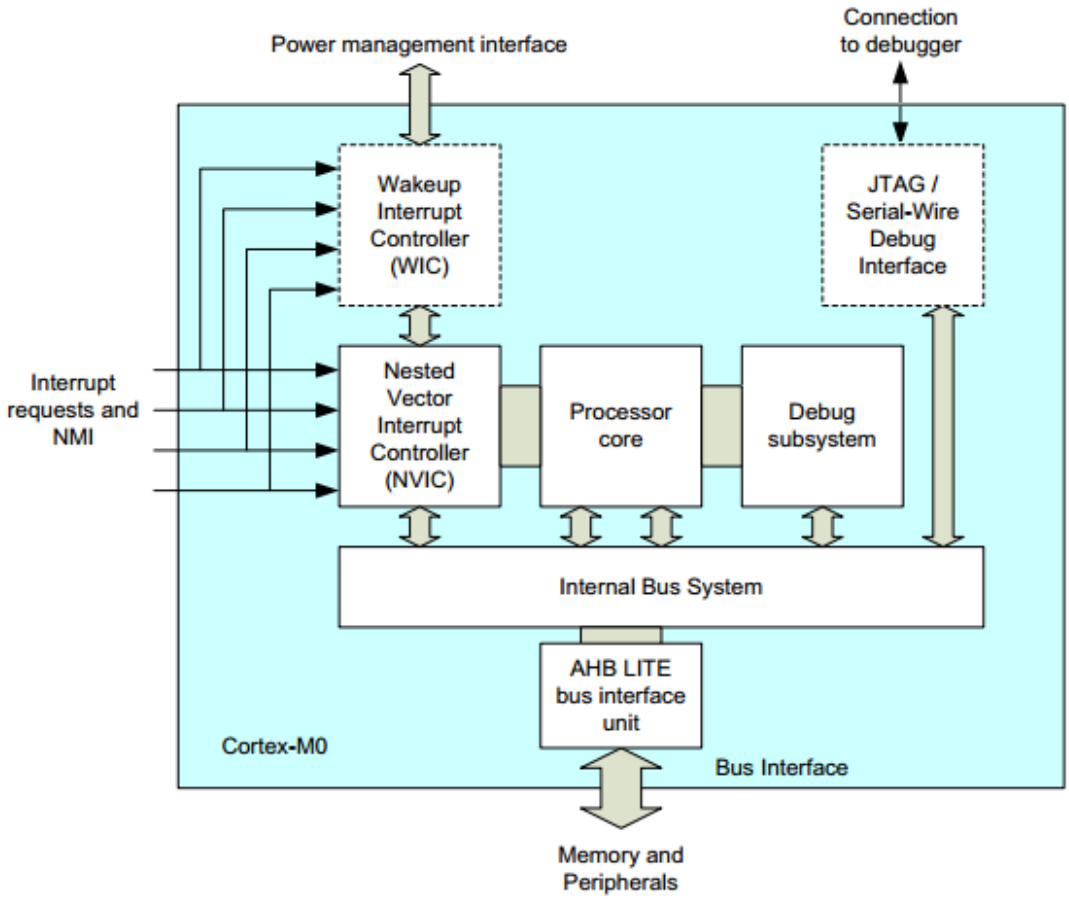
Procesor (přesněji řečeno jádro procesoru) byl navržen, aby uspokojil potřeby dnešních nízkovýkonových mikrokontrolérů a signálových zařízení, u kterých je požadavek na nízkou cenu, možnost použití analogových i digitálních senzorů a zvýšenou konektivitu, což znamená použití sběrnic jako USB, Ethernet, Bluetooth atd. Procesor byl uveden v roce 2009.

Nízkovýkonovost úzce souvisí s maximalizací efektivity, což znamená minimalizace počtu hradel a zlepšování poměru operací za sekundu na megahertz. Na menším počtu hradel totiž vznikají menší ztráty, jak pracovní, tak klidové, a při vyšším počtu operací za sekundu se zkrátí doba práce a zvýší trvání klidového režimu.

Výpočetní efektivita se udává v DMIPS/MHz, kde DMIPS je zkratka pro *Dhrystone milions of instructions per second*. DMIPS se získá testem Dhrystone, při kterém procesor opakuje výpočet speciálního matematického problému s celými čísly (plně využívá ALU). Měřítko na megahertz se pak zavádí pro lepší srovnání, protože frekvence mikroprocesoru může být měněna. Pro ARM Cortex-M0 v minimální konfiguraci se uvádí, že je sestaven z 12000 hradel a má efektivitu 0,9 DMIPS/MHz, dokáže údajně provést součin 32-bitových celých čísel v jenom cyklu. Vyrábí se technologií 90 nm.

Procesor je také vybaven pokročilým správcem přerušení NVIC *(nested vectored interrupt controller)*, který dokáže zpracovávat rozvětvená přerušení s různými prioritami při současné minimalizaci latence přerušení, to znamená trvání ukládání a obnovení kontextu. Tento správce

přerušení je důležitý pro princip minimalizace pracovního času, kdy procesor aktivně vykonává operace, jen když jsou k dispozici data, k čemuž se užívá vnějších přerušení. Rozvržení vnitřní struktury procesoru zobrazuje obrázek 1.



Obr. 1: Vnitřní struktura Cortex-M0+ [1]

Procesor využívá von Neumannovu architekturu (instrukční a operační paměť jsou sdruženy na jedné datové sběrnici) a má redukovanou instrukční sadu (je typu RISC), nazývající se Thumb, která obsahuje 56 instrukcí. Je optimalizovaná na tzv. vysokou kódovou hustotu, neboť zahrnuje 16-bitové i 32-bitové instrukce, kdy se v základě využívají ty 16-bitové, a v případě, že nestačí pro danou operaci, využije se 32-bitová instrukce. Sada Thumb má oddělené instrukce pro zápis a čtení paměti, a pro logické a aritmetické operace využívající registry. Přestože procesor využívá principu pipeline s 3 fázemi, časování instrukcí a přerušení jsou plně deterministická. Další výhodou procesoru je malý rozměr umožněný malým počtem hradel, díky čemuž může v různých aplikacích nahradit 8-bitový mikroprocesor.

Volitelnou součástí procesoru je správce budicího přerušení WIC *(wakeup interrupt controller)*. Používá se u mikrokontrolérů zaměřených na extrémně nízký odběr, protože ty mohou

přejít do režimu spánku s vypnutím většiny periferií a součástí. WIC pak maskuje přerušení přicházející do NVIC, protože je procesor stejně nedokáže obsloužit, protože nemůže vykonat jejich obslužné rutiny. Sám WIC může z vnějšku přijmout speciální přerušení, po němž informuje součást správy energie, která vrátí procesor do režimu aktivity a ten už může vzápětí po odmaskování přerušení NVIC vykonávat příslušné obslužné rutiny.

Procesor ARM Cortex-M0+ kromě role jediného mikroprocesoru MCU také v praxi nachází využití jakožto sekundární procesor nízkovýkonových zařízení, která v režimu stand-by pro úsporu energie nepoužívají svůj hlavní výkonný procesor, dokud to není nutné, a namísto toho nechají sekundární procesor spravovat informace z vnějšku a případně zpracovávat zanedbatelné objemy dat s tím, že v případě potřeby předá opět řízení procesoru primárnímu.

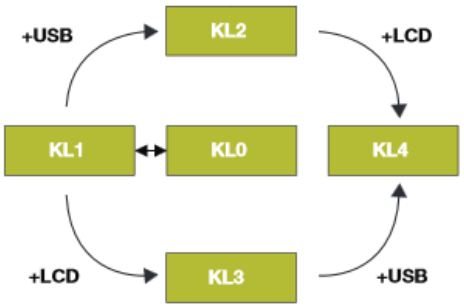
3.2 Kategorie řady Kinetis L

Všechny MCU v této řadě mají společné rysy, kategorie se pak liší v účelu a zároveň se stupňuje výkon mikroprocesoru s rostoucím číslem kategorie. Jednotliví členové kategorií jsou pak také očíslovaní a v rámci kategorie se taktéž stupňují parametry.
Společné rysy:

- Procesor s jádrem Cortex-M0+
- Takt procesoru 45 MHz
- Několik režimů úspory energie LLWU
- Odpojitelnost periferií od zdroje hodinového signálu
- Napěťová logika 3,3 V
- Teplotní rozsah −40 °C až 105 °C
- Nonvolatilní paměť Flash technologie TFS *(thin film storage)* 90 nm
- Operační paměť statického typu
- Ochranné principy v paměti
- Přímý přístup do paměti z periferií DMA *(direct memory access)*
- A/D převodník s rozlišením 12 nebo 16 bitů
- D/A převodník s rozlišením 12 bitů
- Rychlý komparátor
- Asynchronní sběrnice UART
- Synchronní sběrnice I^2C a SPI
- Hodiny reálného času

- Nízkoodběrové časovače, včetně 16-bitových TPM *(timer pulse-width modulator)* modulů
- Časovač periodického přerušení s limitací 32 bitů

Většina produktů užívá z hlediska rozložení stejnou DPS, která napodobuje rozložení na mikrokontrolérech Arduino, díky čemuž zajišťuje kompatibilitu s jejich tzv. shieldy. Taktéž většina desek obsahuje malý dotykový panel a akcelerometr. Obrázek 2 popisuje vzájemný vztah kategorií Kinetis L, které jsou níže detailněji popsány. Srovnání jednotlivých produktů pak zobrazuje obrázek 3.



Obr. 2: Vztah kategorií Kinetis L [2]

3.2.1 Kategorie KL0x

Je to nejnižší kategorie, která se zaměřuje především na nízkou cenu. Rozložení vývodů na čipu mikroprocesoru je stejná, jako u 8-bitových, tudíž jsou produkty této kategorie nabízeny jako snadná možnost přechodu do této vyšší sféry. Samozřejmě s touto vlastností souvisí i malý rozměr pouzdra, a to 4×4 mm. Nejnižší z produktů má výjimku ze společných znaků, neboť nemá D/A převodník, LLWU, ani DMA.

3.2.2 Kategorie KL1x

Jedná se o vyšší verzi předchozí kategorie, protože výrazně přidává na operační i programové paměti, výrazně také zvedá počet vstupně výstupních vývodů. Taky u některých produktů nahrazuje 12-bitový A/D převodník 16-bitovým.

3.2.3 Kategorie KL2x

Oproti předchozí kategorie se příliš neliší, zavádí však zásadní vlastnost, a to USB OTG. Zahrnují totiž řadič pro tuto sběrnici, díky čemuž lze k MCU připojit různá zařízení, např. flash disky.

3.2.4 Kategorie KL3x

Navazuje na kategorii KL1x. Opět se oproti této kategorii příliš neliší, ale procesory zahrnují řadič pro LCD displej, a to až s 375 segmenty. Kvůli LCD vlastnosti je zvýšený počet vývodů. Desky MCU pak přímo zahrnují malý LCD displej.

3.2.5 Kategorie KL4x

Jedná se o kombinaci kategorií KL2x a KL3x v nejvyšší HW konfiguraci. Kombinací je myšleno, že zahrnuje USB OTG i LCD zároveň.

Optional Features											
				Key Features							
Family	Flash	SRAM	Pin Count	USB OTG	Seg LCD	DMA	LLWU	ADC	DAC	I²S	TSI
KL46	128–256 KB	16–32 KB	64–121	√	√	√	√	16-bit	12-bit	√	√
KL36	64–256 KB	8–32 KB	64–121		√	√	√	16-bit	12-bit	√	√
KL34	64 KB	8 KB	64–00		√	√	√	12-bit			
KL26	32-256 KB	4–32 KB	32–121	√		√	√	16-bit	12-bit	√	√
KL25	32–128 KB	4–16 KB	32–80	√		√	√	16-bit	12-bit		√
KL24	32–64 KB	4–8 KB	32–80	√		√	√	12-bit			
KL16	32–256 KB	4-32 KB	32-64			√	√	16-bit	12-bit	√	√
KL15	32–128 KB	4–16 KB	32–80			√	√	16-bit	12-bit		√
KL14	32–64 KB	4–8 KB	32–80			√	√	12-bit			
KL05	8–32 KB	1–4 KB	24–48			√	√	12-bit	12-bit		√
KL04	8–32 KB	1–4 KB	24–48			√	√	12-bit			
KL02	8–32 KB	1–4 KB	16–32					12-bit			

[1] Feature not available on CSP packages
[2] For KL02, use software to support

Obr. 3: Srovnání produktů jednotlivých kategorií Kinetis L [2]

4 Algoritmy pro nízkoodběrové systémy

4.1 Definice pojmu a vytyčení kontextu

Nejprve je nutné stanovit si pojem nízkoodběrovosti, aby bylo možné vytyčit nějaké cíle, především kvantitativního charakteru. Tento pojem není nikde jasně definován, proto nezbývá jiná možnost, než pokusit se stanovit jej logickou indukcí.
Budiž zavedeno následující tvrzení.

Definice 1 *Nízkoodběrovost je vlastnost zařízení, kdy je z jeho zdroje energie odebírán výkon dostatečně malý vzhledem k parametrům tohoto zdroje, s přihlédnutím na účel zařízení.*

Poznámka 1 Zařízením se rozumí MCU s externími senzory, indikátory, komunikátory, úložišti apod.

Ve sféře mikrokontrolérů není zapotřebí se nízkoodběrovostí vůbec zabývat, pokud je zařízení napájeno ze síťového zdroje. V takovém případě máme k dispozici s nadsázkou řečeno neomezenou energii. Běžné síťové adaptéry mívají maximální proud alespoň 500 mA, což pro provoz MCU musí stačit.

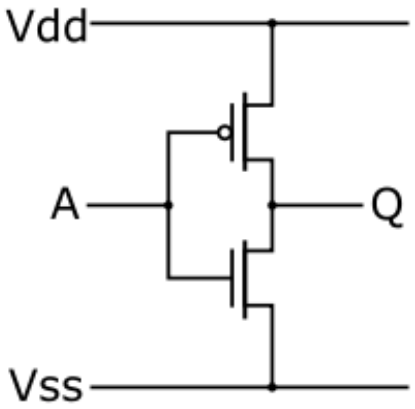
Jiná situace nastává, když je zdroj elektrochemický, to znamená baterie či akumulátor. Tehdy je nutné, aby z tohoto zdroje nebyl odebírán proud, pro který by zdroj nestačil svojí tvrdostí, nebo takový, který by způsobil příliš rychlé vybití, což by bylo přítěží pro koncového uživatele. Překročení tvrdosti elektrochemického zdroje je dokonce přímo patologická záležitost, neboť při tom dochází k poklesu napětí, což může vést k znefunkčnění samotného zařízení, a není dobré pro zdroj jako takový. V případě akumulátorů je situace snazší, protože ty obecně poskytují vyšší proudy, než baterie (mají nižší vnitřní odpor) a také u nich dochází k menšímu poklesu napětí s klesajícím nabitím. Je také skoro zbytečné zmiňovat, že obecně mají širokou škálu vyšších kapacit.
Pro určitou kvantifikaci, běžné alkalické baterie dokážou stabilně poskytovat proudy v řádu jednotek mA, akumulátory typu Li-Ion a Li-Pol pak v řádu desítek až stovek mA. Je-li uvažováno zařízení výše popsaného druhu mající jako zdroj běžný akumulátor s kapacitou řádově 1000 mAh, neměl by průměrný odběr překračovat 1 mA. Ideální by byla hladina v desítkách µA.

4.2 Popis základních algoritmů

Nyní je nasnadě pojmenovat nějaké základní algoritmy nízkoodběrovosti, odůvodnit je a naznačit jejich praktické použití.

4.2.1 Řízení frekvence

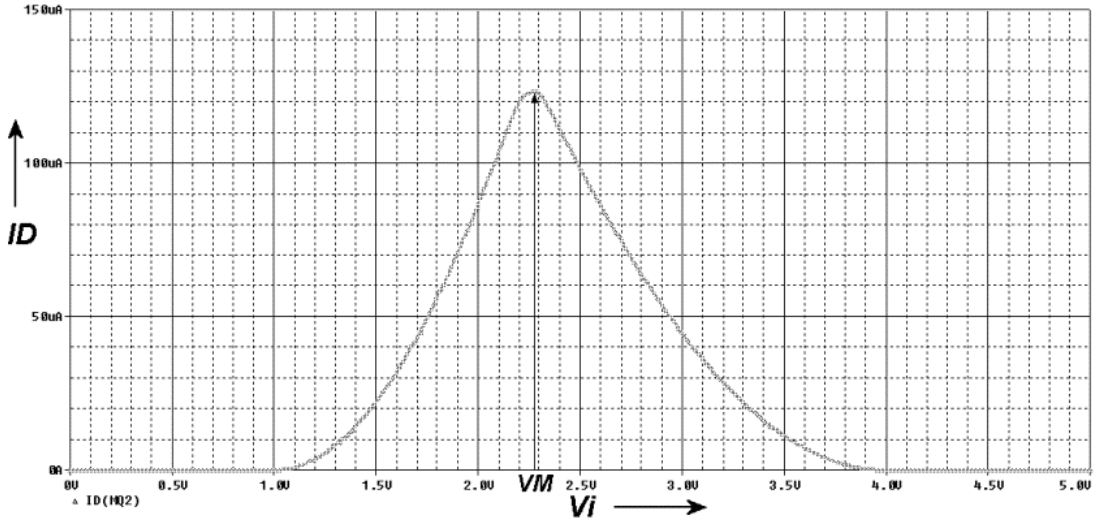
Frekvence je naprosto zásadním faktorem v odběru MCU, existuje přímá úměra mezi frekvencí a příkonem. Tento fakt není nijak překvapivý, zřejmě by ho akceptoval i laik, který má trochu zkušenosti s počítači, avšak stěží by ho dokázal vysvětlit. Tento vztah totiž plyne z obecných zákonů elektrotechniky a pak především z konstrukce hradel.
V elektrotechnice obecně existuje pravidlo, že při vysokých frekvencích od řádu MHz se mění chování soustav a selhávají zjednodušené modely, neboť se začínají značně projevovat jevy jinak zanedbatelné. Projevuje se indukčnost vodičů a vzájemná kapacita mezi jednotlivými vodiči, vytváří se vazby, které jsou zdrojem ztrát. V případě frekvencí v GHz se dokonce i cesty na DPS stávají elektricky dlouhým vedením a musí se na ně uplatňovat model s distribuovanými parametry, to však není případ řádu MHz, neboť tam je vlnová délka stále v řádu metrů.
Největší vliv má však provedení samotného procesoru. Procesor totiž není nic jiného, než soustava CMOS hradel tvořených integrovanými MOSFET tranzistory. V technologii CMOS se hojně využívá komplementárních dvojic N a P typů tranzistoru MOSFET. Velmi dobře se to dá vysvětlit na nejzákladnějším hradle NOT, které je znázorněno na obrázku 4.



Obr. 4: CMOS NOT hradlo [5]

V případě, že je na vstup přivedena 0, je horní P tranzistor otevřen a dolní N tranzistor zavřen, tedy je na výstupu 1. Pro přivedenou 1 je situace přesně inverzní. Jedná se sice o obvody s diskrétním signálem, který nabývá hodnot z vysokého nebo nízkého rozsahu, jenže žádný signál nedokáže nemá nekonečně krátký náběh, signály reálně nejsou obdélníkové, ale lichoběžníkové. Když se napětí při růstu pohybuje v zakázaném pásmu, jsou oba tranzistory ve stejném bodě charakteristiky a jsou částečně otevřené. Tím vytváří cestu sníženého odporu, přes který se uzavírá obvod od zdroje a tento protékající proud je čistě ztrátový. Plyne z toho odběrová charakteristika hradla, znázorněná na obrázku 5, ze které je pozorovatelné, že tento ztrátový proud je mnohonásobně vyšší, než klidový pracovní. U ostatních hradel, z kterých je nejdůležitější

hradlo XOR jakožto základ CMOS, je situace analogická. Z předchozího vyplývá závislost na frekvenci. Čím je frekvence vyšší, tím častěji za sekundu dojde ke ztrátovému přeběhu hradla. Závěr tohoto je takový, že snížení frekvence vede ke snížení odběru.



Obr. 5: Odběrová charakteristika CMOS NOT hradla [5]

Záleží na konkrétním MCU, jaké možnosti řízení frekvence poskytuje. Lepší MCU mají interní i externí krystalový oscilátor a za ním násobičku frekvence, často kombinovanou ještě s děličkou, aby nabízely více poměrů. Některé pak umožňují frekvenci oscilátoru změnit za běhu, jiné ne. Volba základní frekvence si žádá rozvahu, neboť její snížení bohužel znamená i snížení výkonu. Pokud však aplikace nezahrnuje výpočetně náročné kroky, jako je různá analýza signálu stojící na iteračních metodách, je možné ji snížit oproti maximální frekvenci několikanásobně.
Pokud MCU umožňuje měnit frekvenci za běhu, vhodným algoritmem pak bude upravovat ji podle náročnosti úloh; zvýšit ji, když probíhá zpracování dat nebo komunikace, mírně snížit v případě pouhé rutinní činnosti, a poté ji minimalizovat při nečinnosti.

4.2.2 Řízení aktivity periferií

Periferiemi je zde míněn vstupně výstupní modul GPIO. Mikroprocesory zaměřené na nízkoodběrovost jsou vybaveny principem *clock gating*, což znamená, že hodinový signál ze sběrnice je na periferii přiveden přes bránu, která ho může odpojit. Zpravidla má tato brána vlastní registr, ve kterém jednotlivé bity ovládají přívod k jednotlivým portům GPIO. Porty sledují změnu svých registrů právě s taktem sběrnice, při čem spotřebovávají energii. Odpojení od hodin tomu zamezí. Kvůli tomuto algoritmu je nutné dobře zvolit rozložení použitých pinů v rámci portů, ve smyslu sdružovat je, aby bylo možné port odpojit bez ovlivnění pinů, které ovlivněny být ne-

mají. Zde zdrojem problémů mohou být piny používané sběrnicemi, které se nachází na různých portech.

4.2.3 Řízení napájení externí elektroniky

Velmi typické v mnoha aplikacích je to, že jsou k MCU připojena externí aktivní zařízení, především je řeč o senzorech a řadičích. Ty mají vlastní elektroniku, dokonce mají často vlastní mikroprocesor, a tudíž trvale spotřebovávají energii, i když nejsou využívány. Proto může být žádoucí odpojit je od zdroje. To se dá zprostředkovat poměrně snadno, stačí jejich napájení přivést přes tranzistor a ten řídit nějakým pinem z GPIO.

Tento algoritmus nám ale může vyvolat nutnost řešit další časovací úlohu, neboť charakter některých senzorů nemusí umožňovat rychlé obnovení. Některé senzory, především SMART, provádí po spuštění kontrolu, autokalibraci a potřebují nějakou dobu, než jsou schopny poskytovat relevantní výsledky. Čekat ale aktivně na náběh kupříkladu minutu je energeticky drahé, proto je optimálním způsobem senzoru přivést napájení, případně provést aktivační příkazy, a poté nějakým způsobem s využitím ostatním algoritmů šetřit energii a nastavit nízkoodběrový časovač pro ukončení tohoto režimu v době, kdy bude senzor připraven.

4.2.4 Přechod do úsporného režimu

Mikroprocesory zaměřené na nízkoodběrovost mají speciální režimy redukované činnosti, při kterých využívají zlomek příkonu. Ty lepší mají těchto režimů více, ve smyslu více stupňů. V základních rysech však mívají srovnatelné 2 úrovně úsporného režimu.

V základě mikroprocesor běží v plném režimu, kdy jsou všechny součásti funkční. V úsporném režimu *(WAIT)* modul správy energie kompletně odstaví jádro mikroprocesoru, které tímto přestane vykovávat program. Nadále však zůstává případná konfigurace *clock gating* pro periferie a správce přerušení je aktivní a může iniciovat probuzení.

V režimu hlubokého spánku *(SLEEP)* modul správy energie odstaví nejen jádro, ale také odřízne od hodinového signálu všechny periferie a deaktivuje správce přerušení. V běhu zůstává jen nutné minimum, a to součást udržující hodnoty registrů a hlídající integritu. Probuzení je podmíněno speciálním vnějším přerušením v součásti správce energie, může být řeč o LLWU.

Možnost užívat tyto režimy může být podmíněna určitým nastavením časování procesoru. Konkrétní režim je potřeba volit podle okolností a vhodně načasovat jeho spuštění i ukončení. Chyba ve zdroji přerušení by mohla zapříčinit setrvání procesoru v režimu spánku tehdy, kdy už je od něj vyžadována činnost. Ze všech algoritmů bývá přechod do úsporných režimů ten nejefektivnější, v nejlepším případě se odebírané proudy redukují na jednotky µA, v extrémním případě na stovky nA. Nevýhoda však je, že procesor nemůže nic vykonávat.

4.3 Celková metodika

Ve výsledném systému je samozřejmě nutné výše popsané algoritmy aplikovat společně. Orientačně se dá tato problematika shrnout v bodech:

- Nastavit základní frekvenci procesoru co nejnižší s přihlédnutím k povaze převládající činnosti.
- Pokud je to možné, dále snižovat frekvenci v případě vykonávání výpočetně nenáročných úloh a jen při potřebě vykonání těch náročných ji dočasně zvýšit.
- Odpojit externí elektroniku od napájení, když není využívána. U senzorů, které nabíhají delší dobu, je možné navrátit jim napájení s předstihem a na náběh čekat v úsporném režimu.
- Chytře sdružovat piny v rámci portů a porty odpojovat od zdroje hodinového signálu.
- V případě nečinnosti co nejdříve přecházet do úsporného režimu. Typ režimu musí být zvolen s přihlédnutím k předpokládané době nečinnosti a akutnosti následně příchozích úloh.

5 Návrh řešení systému

Zadáním práce je implementovat RFID technologii pro nízkoodběrová vestavná zařízení. Po dohodě s vedoucím práce bylo stanoveno, že se bude jednat o měřicí zařízení pro dlouhodobá pomalá měření, a že bude použit standart NFC, který vychází z technologie RFID. Díky použití NFC bude moct být využit mobilní telefon.
Z tohoto plyne nutnost zvolit jednotlivé součásti daného zařízení, jak po stránce HW, tak SW.

5.1 Řídicí systém

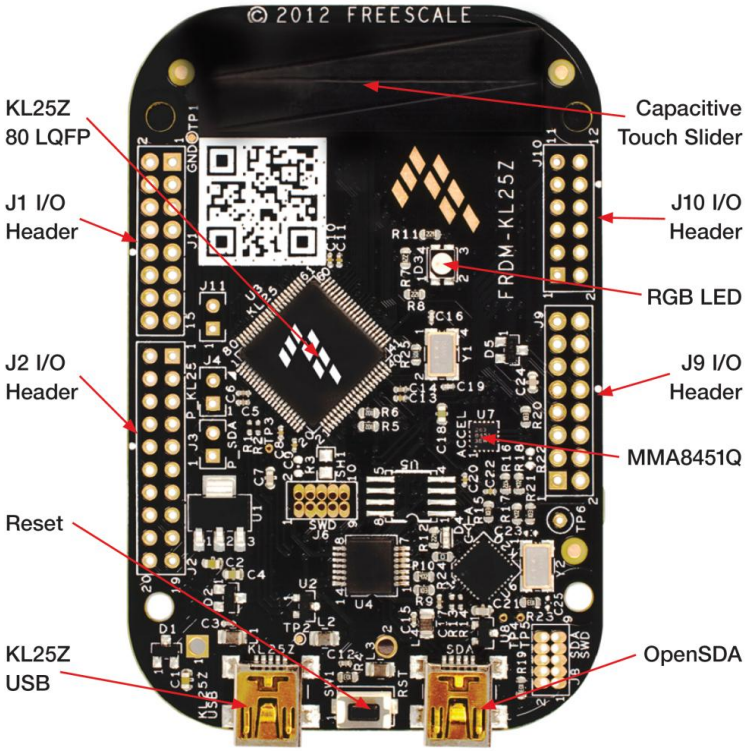
Přirozeně musí být zařízení řízeno nějakým procesorem, jako přirozená volba se jeví mikrokontrolér vyšší kategorie. Nezbytně musí být vybaven základními sběrnicemi jako UART a I^2C a musí zahrnovat A/D převodník; tyto požadavky je snadné splnit. Je také velmi vhodné, aby deska MCU měla nějaký USB port menšího rozměru, tzn. Mini-USB nebo Micro-USB.

Snad nejdůležitější je pak náročnější požadavek na nízkoodběrovost MCU, musí být na tuto vlastnost specializováno. Tím se výběr zužuje. Na základě domluvy s vedoucím byl zvolen mikrokontrolér FRDM-KL25Z. Ten patří do série Freescale Kinetis L, která je zaměřená na nízkoodběrovost a i nízkou cenu. Je postaven na procesoru ARM Cortex M0+, který byl popsán v teoretické části této práce.
Parametry tohoto konkrétního procesoru:

- Maximální frekvence 45 MHz
- Programová paměť 128 kB Flash
- Operační paměť statického typu 16 kB
- GPIO o 80 pinech
- A/D převodník s postupnou aproximací s rozlišením 16 bitů
- D/A převodník s rozlišením 12 bitů
- Mechanismus LLWU
- UART, I^2C , SPI
- Podpora USB OTG
- OpenSDA debugger

Mikrokontrolér je navíc připraven přímo na napájení z baterie (ideálně CR2032, 3 V), je vybavena napěťovým regulátorem. Deska je také vybavena malým kapacitním dotekovým panelem, akcelerometrem a RGB LED diodou. PCB zahrnuje 2 Mini-USB porty, jeden pro USB OTG, druhý pro OpenSDA, přes které probíhá spojení s počítačem. Také má přirozeně resetovací

tlačítko. Na druhou stranu PCB postrádá dutinkové lišty, ty je zapotřebí připájet svépomocí. Mikrokontrolér je vyobrazen na obrázku 6.



Obr. 6: Vyobrazení FRDM-KL25Z s popisy [2]

Vývoj programu bude prováděn v klasickém jazyce C, ve vývojovém studiu CodeWarrior od samotné firmy Freescale. Bude použito nástroje Processor Expert, který zjednoduší práci s registry procesoru. Bude užito nízkoodběrových algoritmů popsaných v teoretické části práce. Frekvence bude nastavena předem, protože procesor neumožňuje měnit ji za běhu. Avšak podporuje úsporné režimy, má právě režimy WAIT, SLEEP a STOP, které budou základem nízkoodběrovosti, neboť bude snaha vykonávat potřebné operace v co nejkratším čase a ihned po nich přecházet do těchto úsporných režimů, kdy potřebná budicí přerušení poskytne NFC zařízení a hodiny reálného času. Hodiny reálného času také poskytnou potřebné časové údaje pro výstup z měření, díky čemuž nebude nutné čas žádným jiným způsobem počítat.

5.2 RFID/NFC zařízení

Zde se nabízí 2 možnosti. Tou první je přímý komunikátor, který by propojil telefon s MCU v reálném čase. Kvůli tomuto charakteru by zřejmě musel být k MCU připojen nějakou časově nedeterministickou sběrnicí jako je UART, nebo v případě časově deterministické sběrnice (jako I^2C , SPI) by tato sběrnice musela být stále aktivní, což by bylo neúsporné, nebo nabízet nějaké

notifikační přerušení. Druhá možnost je použít nepřímý komunikátor s vlastní pamětí, něco jako výměník dat, kam by data mohla vždy přicházet jen z jedné strany a poté být z druhé vyzvednuta.

Pro tuto práci byla zvolena druhá možnost na základě konzultace s vedoucím práce, neboť ten obdržel od firmy STMicroelectronics na prezentační akci vývojový kit pro NFC, protože firma ST se snaží v této oblasti o rozvoj. Kit má označení X-NUCLEO-NFC01A1, jedná se o desku (DPS), která je osazená čipem M24SR, integrovanou anténou, dutinkovými lištami s kontakty ze spodní strany a 3 LED diodami. Deska odpovídá rozměru shieldu pro Arduino UNO, primárně je však určena pro MCU od ST. Deska je vyobrazena na obrázku 7.

Obr. 7: Vyobrazení X-NUCLEO-NFC01A1

5.2.1 Čip M24SR

Čip M24SR je složen z několika součástí. Jeho základem je EEPROM paměť kapacity 8 kB, ke které jsou připojeny 2 komunikační součásti. První je I^2C rozhraní, kde čip vystupuje jako slave, a druhá je NFC rozhraní připojené k anténě. V rámci NFC čip naplňuje specifikace *ISO/IEC 14443*

typ A a *NFC Forum typ 4*, přijímá komunikaci na krátkou vzdálenost s frekvencí 13,56 MHz. Pro přístup přes NFC není nutno připojit napájení, anténa slouží jako zdroj energie po dobu komunikace.

Dále má čip 1 vstupní a 1 výstupní pin. Vstupní je označen *RF disable* a při log. 1 čip zablokuje anténu, takže nebude aktivními zařízeními ani rozpoznána.
Výstupní je označen GPO a jeho účelem je notifikovat vnější MCU o aktivitě ze strany NFC. Má několik režimů činnosti, může být manuálně ovládaný, může automaticky reflektovat přiblížení telefonu k anténě apod.

Smyslem komunikace je zapisovat a číst z paměti. Čip má jakýsi semafor, který znemožňuje přistupovat do paměti najednou z obou stran. Obě strany nejsou rovnocenné, přes NFC nelze některé kroky vůbec provést, např. nastavovat režim GPO. Ze strany I^2C je navíc možné blokovat funkci NFC a dokonce násilně ukončit právě probíhající komunikaci přes NFC.

Čip má také systém zabezpečení, zavádí 3 hesla délky 128 bitů, a to *heslo pro zápis*, *heslo pro čtení* a *I^2C heslo*, všechna se dají měnit a jsou volitelná.

- Heslo pro čtení opravňuje kteroukoliv stranu pouze ke čtení.
- Heslo pro zápis opravňuje kteroukoliv stranu ke čtení, zápisu a změnu obou dosud zmíněných hesel.
- I^2C heslo může být zadáno jen přes I^2C a udělí MCU maximální práva, díky kterým nepotřebuje znát ostatní 2 hesla a může libovolně zasahovat do paměti a měnit hesla. Ztráta I^2C hesla znamená v případě jeho aktivnosti trvalou degradaci čipu, protože neexistuje žádný způsob, jak ho zjistit.

EEPROM paměť se dělí na 3 úseky nazývané jako soubory, a to *System*, *CC* a *NDEF*.

- V System jsou uloženy základní informace a nastavení, většina je nezměnitelná, několik z nich lze změnit přes I^2C .
- CC obsahuje taktéž několik informací o čipu a o jeho zabezpečení, přístup do něj se neprovádí přímo, ale přes speciální příkazy.
- NDEF je pak samotným úložištěm, kde první 2 bajty obsahují číslo udávající počet bajtů v NDEF souboru obsazených daty mínus ony 2 počáteční bajty. První 2 bajty se nemění samovolně, uživatel čipu jej musí vypočítávat a správně nastavovat, neboť čip nedovoluje zapisovat za hranici vytyčenou těmito 2 bajty.

Čip zavádí vlastní komunikační protokol, kdy je součástí každého rámce kontrolní hash CRC-16 algoritmu. Bez správného kontrolního součtu čip odmítá přijímat příkazy. Čip definuje pro své řízení řadu příkazů, které zahrnují inicializaci, čtení z paměti, zápis do paměti, autorizaci, změnu hesla, aktivaci hesla, deaktivaci hesla, ovládání GPO (jen v manuálním režimu), vyslání

přerušení na GPO (jen v přerušovacím režimu) a další, které ani nebudou v této práci užity. Při přístupu přes I^2C je nutno poskytnout desce napájení, a to o napětí 3,3 V.

5.3 Měření

Bude zajištěno senzory a snímači připojenými přímo k FRDM-KL25Z. Měření jednoduchého napěťového charakteru bude realizováno připojením k internímu A/D převodníku, který díky svému rozlišení 16 bitů dokáže poskytnout dobrou přesnost. Pro složitější měření budou použity hotové senzory, které budou mít nejvhodněji výstup na I^2C sběrnici, což nepředstavuje žádné obtíže navíc, tedy kromě nutnosti prozkoumat příkazy pro ovládání senzoru. Časové údaje a taktéž načasování pro vzorkování poskytnou hodiny reálného času, neboť cílem jsou měření pomalá. Senzorů bude zřejmě více pro více měřených veličin a vždy bude připojen jeden přes vhodný konektor, možná JTAG.

5.4 Využití mobilního telefonu

Tato možnost přidá celému projektu na atraktivitě, protože využití tzv. chytrých telefonů je dnes moderním prvkem. Existuje několik různých operačních systémů, které se od sebe liší ve způsobu tvorby aplikací. Volba je dosti jednoduchá, protože operační systém Android jednoznačně dominuje na trhu a výhledově se žádná změna nechystá.

Operační systém Android má mnoho výhod, je značně otevřený a společnost Google, která OS vyvíjí, má přátelskou politiku vůči vývojářům a snaží se jim co nejvíce usnadnit jejich činnost. Kernel OS je postaven na Linuxu a poskytuje vysokou aplikační vrstvu v jazyce Java. To je jazyk vysoký, ve kterém vývojář nemusí řešit přímý styk s operačním systémem, čímž se práce zjednodušuje. Není to jazyk bohužel tak pokročilý, jako je C#, avšak postačující. Google poskytuje přímo specializované nástroje pro vývoj aplikací, a to vývojové studio Android Studio.

Rozvržení stránek se tvoří v XML formátu a jedná se v základě o relativní rozložení. Grafická stránka je dosti zjednodušená. Pro použití NFC Android obsahuje příslušné API, které zavádí třídy pro správu připojení, zjišťování informací o druhé straně a odesílání příkazů s přijetím následné odpovědi. Samozřejmě nutností je, aby telefon byl vybaven NFC komunikátorem, což však v dnešní době není problém. Tato technologie se bouřlivě rozvíjí a všechny vlajkové lodi i nadprůměrné telefony všech výrobců jsou tímto běžně vybaveny.

5.5 Zbylé problémy

Dlouhodobé měření si žádá přesné časové údaje o každém naměřeném vzorku, takže MCU potřebuje znát přesný čas. Avšak při každém restartu tento údaj ztrácí a navíc nedokáže reflektovat časová pásma. Proto je žádoucí, aby uživatel mohl nastavit svůj aktuální čas. Jenže spojení mezi

MCU a telefonem je nepřímé, takže než by došlo k vyhodnocení dat, mohlo by uběhnout několik sekund. Proto je nejschůdnější cestou vytvoření jednoduché aplikace pro Windows používající framework .NET, napsané v neobyčejně příjemném jazyce C# v prostředí Microsoft Visual Studio, která by umožnila přes UART emulovaný na USB předat systémový čas, nebo přímo přesný čas z nějakého časového serveru, např. od amerického institutu pro normalizaci a technologii (NIST).

5.6 Celkový způsob použití

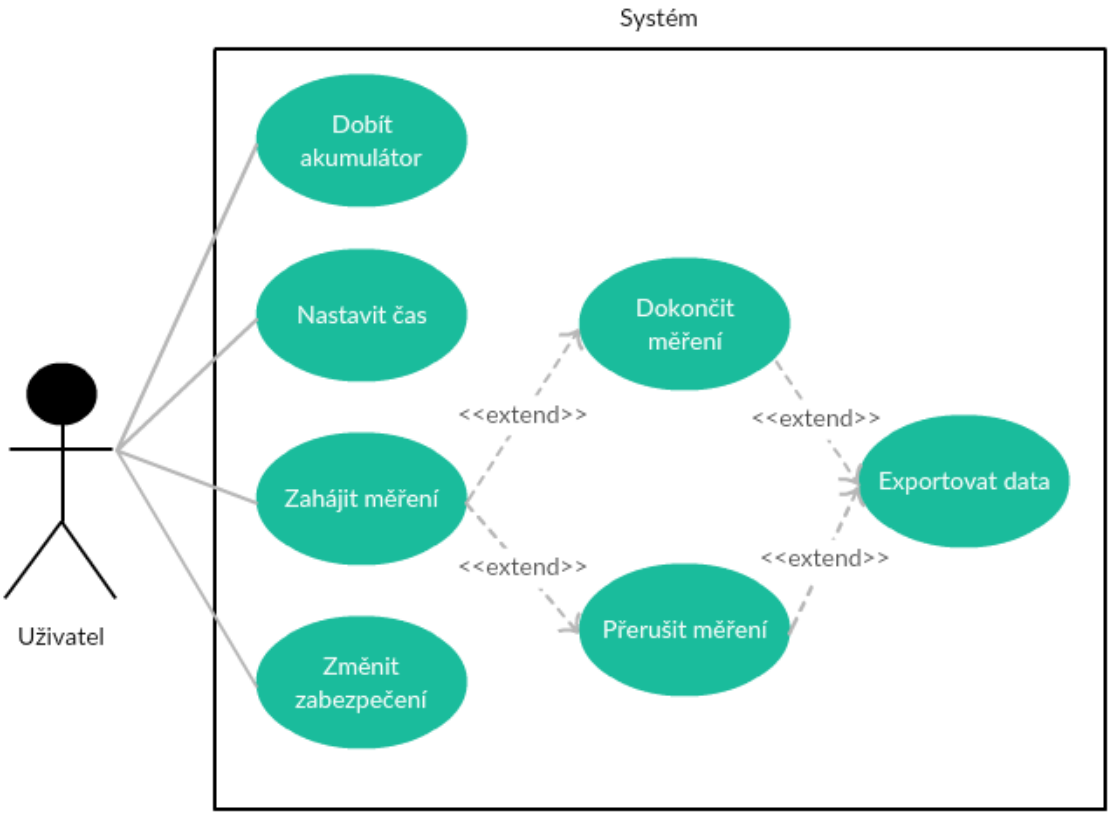
Myšlenkou je poskytnout uživateli zařízení pro dlouhodobá pomalá měření ovládané telefonem. Uživatel po získání produktu zařízení spojí přes USB s počítačem a pomocí aplikace pro Windows nakonfiguruje správný čas podle svého časového pásma. Rovněž přes USB dobije akumulátor zařízení.

Poté umístí zařízení na místo měření a přes přiložený konektor k němu připojí senzor, který si vybral z nabízených. Na svém telefonu s OS Android spustí aplikaci, kde si případně nastaví zabezpečení dat a především si v průvodci vybere správné měření s žádanou konfigurací, a to typ měření, čas začátku měření, periodu měření, čas konce měření nebo počet vzorků. Poté přiloží telefon k zařízení a zapíše do něj svůj požadavek.

Díky přerušení z GPO se probudí MCU z úsporného režimu, vložené informace přečte, vyhodnotí a načasuje si podle toho hodiny reálného času. Ve správný čas zahájí měření a každé další měření si načasuje stejně, aby mezi nimi mohl být v úsporném režimu. Díky tomu bude moct měřit po velmi dlouhou dobu. Po skončení rozbliká indikační LED diodu nebo případně nějakým jiným způsobem upozorní uživatele, že si má vyzvednout naměřená data.

Uživatel po zjištění skončení měření k zařízení opět přijde a v aplikaci si ověří, že je vše v pořádku. Poté si výsledky měření nechá přenést ze zařízení do aplikace, která z dat vygeneruje CSV soubor. Ten bude uživatel moct vložit do jakéhokoliv SW, jako je MATLAB, Microsoft Excel, LabVIEW a mnoho dalších.

Na konci realizaci popsaných funkcionalit stojí za zvážení vytvoření online systému, kam by uživatel mohl svá data přes aplikaci odeslat a poté si je stáhnout přes počítač, a to buďto přímo ve Windows aplikaci, nebo např. přes odkaz na webové stránky poslaný na emailovou adresu, což je více pravděpodobná možnost. Takový princip by uživateli příjemně zjednodušil transfer dat do počítače. Možnosti přístupu k systému vizualizuje diagram 8.



Obr. 8: Use case diagram systému

6 Realizace řešení

Navržené řešení problému bylo realizováno v souladu s předešlou kapitolou, byly splněny vytyčené náležitosti. Při realizaci se však vyskytly některé problémy či zvláštnosti, které si vyžádaly přehodnocení a trochu jiné pojetí či složitější postup, neobjevil se však žádný problém, který by byl nepřekonatelný.

6.1 Řídicí systém pro MCU

Byl užit zvolený mikrokontrolér FRDM-KL25Z, pro něhož byl vyvinut nejprve obecný program pro základní běh, zahrnující knihovny pro řízení potřebných sběrnic (UART a I^2C) a také měření na A/D převodníku. V další fázi byly implementovány knihovny pro přístup k čipu M24SR. Na závěr byl konečně implementován řídicí systém uplatňující zmíněné algoritmy a zaštiťující celou funkčnost, aby mikrokontrolér mohl reflektovat změny provedené druhou stranou (telefonem).

6.1.1 Základní program

Tento základ slouží jako kostra pro tento projekt a je znovu použitelný pro jakékoliv další. Níže jsou popsány části související s touto prací, ve skutečnosti je totiž rozsáhlejší. Zahrnuje časování (cyklické i blokovací), ovládání RGB LED diody, čtení a zápis na piny GPIO, vnější přerušení z GPIO, hodiny reálného času, řízení UART a I^2C , měření A/D převodníkem.

6.1.1.1 Časování Byla vytvořena knihovna *Timing*. Jednodušším případem časování je blokovací časování neboli aktivní čekání, kdy je úmyslně mařen procesorový čas opakováním prázdné instrukce. To bylo realizováno ve funkci *void Timing_Wait(unsigned long cycles)* pomocí cyklu for, ve kterém je přímo v jazyce symbolických adres zápis prázdné instrukce. Nejedná se o zpoždění se vstupem s časovou jednotkou, avšak slouží k prostému pozdržení, které je mnohde potřebné.

Ovšem je také žádoucí, aby mohly být úseky kódu vykonávány pravidelně přibližně s přesnou periodou. K tomu bylo využito PE komponenty TimerInt, která pomocí časovače (a komparátoru na modulo registru) generuje periodické přerušení každou 1 ms. Její obslužná rutina volá funkci, která počítá počet milisekund přičítáním do proměnné a vypočítává modulo s hodnotami pro setinu, desetinu, čtvrtinu, polovinu a celou sekundu, při jehož rovnosti nule nastaví boolovský příznak indikující, že je vyžadována akce v hlavní smyčce. Bylo by totiž velmi neuvážené provádět rozsáhlejší operace přímo v obslužné rutině, protože by při vzniku dalších přerušení mohlo dojít k definitivnímu zacyklení.

6.1.1.2 Ovládání GPIO

Řeč je o přepnutí pinů vstupně výstupní periferie na režim vstupní či výstupní a zápis či čtení hodnoty. Na nejnižší úrovni se toto provede změnou hodnot v registrech. Nejprve se operací OR přidá do registru *SIM_SCGC5* maska povolující napájení pro příslušný port. Každý pin má svůj registr, kam se nastavuje multiplexer (určuje režim práce pinu, číslo 1 je pro digitální) a může se přidat příznak pro zvýšený odběr. Každý port má pak registr s koncovkou *PDDR* pro nastavení směru pinu (vstupní či výstupní), *PSOR* pro přepnutí do vysoké úrovně, *PCOR* pro reset do nízké úrovně a *PTOR* pro přepnutí do opačné úrovně.

Jednodušší přístup je použít PE komponenty s názvy BitIO, BitsIO a ByteIO. BitIO slouží k ovládání jednoho pinu, BitsIO pro ovládání 1 až 8 pinů a ByteIO pro ovládání celého portu o 8 pinech. Všechny vygenerují velmi jednoduché funkce pro použití. Pro ovládání RGB LED diody byl použit způsob první. Pro blokaci funkce antény byl užit druhý způsob.

Z GPIO lze také získat vnější přerušení, což bylo potřebné, a k tomu byla použita PE komponenta ExtInt, u které se v průvodci nastaví pin a detekovaná hrana. Sama se vytvoří obslužná rutina. Toto bylo užito pro získávání přerušení z GPO pinu modulu, které oznamuje, že u antény je přítomen telefon. Obslužná rutina jen nastavuje příznak, že je nutné vyhodnocení a ukládá aktuální čas, aby mohlo být provedeno se zpožděním poté, co je telefon oddálen.

6.1.1.3 Hodiny reálného času

V tomto projektu jsou nezbytné pro realizaci měření časovaného podle kalendáře. Přímo procesor má modul hodin reálného času integrovaný, softwarově jsou realizovány komponentou TimeDate. Ta si uchovává datum a čas (s předností na setiny sekundy) v podobě 2 struktur, má metody pro jejich přečtení i nastavení. Velmi důležité je, že se dá nastavit alarm na přesný čas, při kterém se vyvolá přerušení. Alarm není nijak podmíněný datem, takže kontrola správnosti data je věcí vlastní implementace. Stejně tak komponenta není nijak nápomocná pro nějaké aditivní operace s časem, proto přičtení k datu bude řešeno vlastním algoritmem později.

6.1.1.4 UART

Jedná se o složitější problém, UART (též označován jako sériová linka) je sběrnice asynchronní, stanice jsou v ní rovnocenné, a proto přijímaní dat není časově deterministické a může nastat v libovolných časech. Proto musí v příjmu dat figurovat buffer dočasné uložení, než budou moct být vyhodnocena. Jako základ pro řešení byla použita nízkoúrovňová PE komponenta Serial_LDD. V průvodci se nastaví parametry sběrnice, které se pak musí shodovat na druhé straně, protože stanice nemají možnost, jak se na parametrech domluvit. Nastavení bylo z většiny ponecháno základní, tzn. blok 8 bit, 1 stop bit a žádná parita. Baud rate byl změněn na 38400 baud. Komponenta má funkci pro odeslání a přijetí dat, a má událost pro příjem a také pro dokončení odesílání.

Nezbytná podpůrná komponenta je importovaná, jedná se o RingBufferUInt8 a obsahuje funkce, které by se u fronty daly čekat (přidání prvku, odebrání, počet).

Funkce komponenty Serial_LDD jsou však nedostačující pro pohodlné používání, je zapotřebí zaobalit je vlastní knihovnou, ta dostala jméno UART. Komponenta zavádí strukturu pro předávání dat a referencí, která je v knihovně napodobena, viz výpis 1. Je totiž nezbytná pro spojení s obslužnými rutinami, které komponenta vygeneruje.

```
typedef struct
{
   LDD_TDeviceData *handle;
   volatile bool isSent;
   uint8_t rxChar;
   uint8_t (*rxPutFct)(uint8_t);
} UART_Desc;
```

Výpis 1: Struktura knihovny UART pro spojení s komponentou [7]

Základní úlohou knihovny je poskytnout funkce pro odeslání znaku a celého řádku, což vychází z podstaty sériové linky. Taktéž musí zajistit obsloužení příjmu dat a jejich uložení do bufferu. Navíc obsahuje funkci pro porovnání obsahu bufferu s řetězcem, aby bylo možné rozpoznávat stanovené příkazy a navíc má prototyp funkce *UARTevaluate*, kterou sama nedefinuje a ponechává to k pozdější implementaci (protože se bude jednat o příliš konkrétní kód), a tuto funkci bude volat když do bufferu příjme celý řádek oddělený zlomem. Relevantní deklarace a prototypy jsou znázorněny ve výpisu 2.

```
void UART_Init();

void UART_SendChar(char ch);
void UART_SendLine(char* str);

char UARTincoming[128];
int UARTincoming_nxtind;
void UARTevaluate(void);
bool UARTincoming_CompareWithString(char* charray);
void UARTincoming_ResetIndex(void);
```

Výpis 2: Veřejné deklarace a prototypy knihovny UART

Knihovna přidává ještě jedno usnadnění, a to jakýsi výstupní buffer, do kterého se přes několik funkcí dají přidat různá data, které se správně zformátují. Takto se dá připravit do jednoho řetězce více dílčích řetězců, znaků a čísel, a poté jen vše najednou odeslat. Příslušné prototypy jsou znázorněny ve výpisu 3.

```
void UARTQueue_Init();
void UARTQueue_Send();
void UARTQueue_AddChar(char ch);
void UARTQueue_AddString(char* string);
void UARTQueue_AddNumber(int number);
```

Výpis 3: Veřejné prototypy pro odchozí frontu v knihovně UART

6.1.1.5 I²C

Situace je o něco jednodušší, než u sběrnice UART. Sice má I²C mnohem složitější linkovou vrstvu, avšak to je věc, kterou vyřeší nízkoúrovňová komponenta I2C_LDD. Co zjednodušuje situace je fakt, že se jedná o sběrnici synchronní a je v tomto projektu používána čistě jako single-master, kdy onen master je právě MCU. Parametry se nastaví v průvodci, musí se hlavně vybrat piny pro SDA a SCL a zvolit vhodná frekvence hodin, bylo vybráno 300 kHz. Stejně jako u UART, jsou užívány funkce komponenty pro odeslání a příjem dat a také události pro příjem a dokončení odesílání. Nyní však vznik události pro příjem bude předvídatelný, protože ve zvoleném režimu je chování časově deterministické.

Funkce komponenty jsou však opět nedostačující a bylo nutné vytvořit knihovnu, nazvanou I2C. Opět pro spojení s komponentou je zapotřebí napodobit strukturu, kterou používá v argumentech, ta zde již nebude popisována kvůli analogii.

Podstatné je, že knihovna I2C poskytuje jednoduché funkce pro nastavení cílové adresy (sběrnice užívá 7-bitovou adresaci), a odeslání a přečtení pole bajtů. Při komunikaci se striktně kontroluje obdržení ACK a komunikace je přerušena, když dojde k selhání. Relevantní prototypy jsou zobrazeny ve výpisu 4.

```
void I2C_Init(void);
void I2C_Deinit(void);

uint8_t I2C_SetSlaveAddress(uint8_t address);

uint8_t I2C_WriteBytes(uint8_t* bytes, uint8_t count);
uint8_t I2C_ReadBytes(uint8_t* bytes, uint8_t count);
```

Výpis 4: Veřejné prototypy knihovny I2C

6.1.1.6 A/D převodník

V tomto projektu je uplatněno měření napětí jakožto výchozí možnost. KL25Z má samozřejmě zabudovaný A/D převodník, a to až s rozlišením 16 bit, pro tento

projekt bylo zvoleno rozlišení 12 bit, které je i tak vysoké. Na výběr je taktéž mezi RSE a diferenciálním režimem, jako vhodnější byl vybrán první. Zvoleny byly 2 kanály, kterým byly přiřazeny piny. Doba převodu vyšla na 3,8 µs.

Všechny výše zmíněné parametry se nastaví v průvodci vysokoúrovňové komponenty ADC. Její funkce nejsou úplně ideální, a proto i nad ní byla stvořena knihovna ADC. Ta je však značně chudá a zavádí jedinou funkci, jejíž prototyp je ve výpisu 5.

```
uint8_t ADC_SingleMeasure(uint8_t channel, uint16_t* measurement_result);
```

Výpis 5: Veřejný prototyp knihovny ADC

6.1.1.7 Podpůrné algoritmy

Kvůli potřebě ukládat do paměti data ze senzorů je nezbytné převádět mezi různými druhy celých čísel, neboť paměť se adresuje po bajtech, avšak už informace z A/D převodníku má 12 bit, a tudíž jí 1 byte nestačí. Převod se provádí pomocí bitových operací, především AND, OR a bitový posun.

Kvůli tomuto byla vytvořena knihovna DataConversions. Ta zároveň obsahuje algoritmy pro převod mezi čísly a znaky (resp. skupinou znaků), které jsou potřebné pro práci s UART, především výstupní fronty. Tento převod se provádí cyklickým výpočtem s pomocí dělení a operace modulo, a také znalosti ASCII tabulky. Vnější prototypy jsou zobrazeny ve výpisu 6.

```
void DataConversions_SplitUINT16toUINT8(uint16_t input, uint8_t* output, bool
    swap);
void DataConversions_CombineUINT8toUINT16(uint8_t input1, uint8_t input2,
    uint16_t* output);
void DataConversions_SplitUINT32ToUINT8(uint16_t input, uint8_t* output);

float DataConversions_CharsToFloat(char* chars, int chars_count);
int DataConversions_CharsToInt(char* chars, int chars_count);
int DataConversions_IntToChars(int number, char* chars);
```

Výpis 6: Veřejné prototypy knihovny DataConversions

6.1.2 Ovládání čipu M24SR

Jak bylo dříve popsáno, čip má privilegovaný přístup přes I^2C . Má stanovenou sadu příkazů, popsanou ve své dokumentaci. Příkazem se rozumí určitý předpis pro rozložení bajtů. Na začátku je vždy několik bajtů indikujících, o jaký příkaz se bude jednat, pak následuje upřesnění

a nakonec, pokud se jedná o práce s daty, už jsou bajty užitečných dat. Určení konkrétních systémových bajtů věc podrobného studia datasheetu.

Velmi důležité je, že pro ověření integrity komunikace se používá CRC-16 a to tak, že na konci příkazu se musí přidat 2 bajty, ve kterých je kontrolní „součet“ ze všech předchozích bajtů. Proto byla v knihovně M24SR, která pokrývá celou práci s čipem, vytvořena funkce pro přepočítání a přidání CRC bajtů do příkazu. Slovo „přidání“ není úplně přesné, neboť ve všech komunikujících funkcích knihovny je užíván způsob, že pole bajtů má ve své velikosti 2 prázdné bajty navíc, kam se právě CRC doplní. Nízký jazyk C totiž lepší řešení neumožňuje.

Vnější prototypy a deklarace použitých enumerátorů jsou zobrazeny na výpisu 7. Všechny funkce budou blíže popsány, neboť jsou důležitou součástí cílů tohoto projektu. Knihovna je úmyslně psána dosti obecně, aby mohla s menšími úpravami být přenesena na jakýkoliv jiný mikrokontrolér nabízející jazyk C. Správné sestavení postupů bylo dosti časově náročné, protože dokumentace čipu není v některých ohledech dosti jednoznačná a hodně poznatků bylo zjištěno experimentálně.

```
//I2C address
#define M24SR_ADDR 0b1010110

//Enumerators
typedef enum
{
   System,
   CC,
   NDEF
} M24SR_Files;

typedef enum
{
   SessionOpen,
   WIP,
   I2CAnswerReady,
   MIP,
   INT,
   StateControl,
   RFBusy
} M24SR_GPO_Modes;
```

```
typedef enum
{
   NDEF_Read,
   NDEF_Write,
   I2C
} M24SR_AccessTypes;

typedef enum
{
   verify,
   change
} M24SR_AccessActions;

typedef enum
{
   protect_read,
   protect_write,
   unprotect_read,
   unprotect_write
} M24SR_SetNDEFProtectionActions;

//Session
bool M24SR_Ready(bool forced);
bool M24SR_EndI2CSession(void);

//Boolean response
bool M24SR_GetResponse(void);

//Security
bool M24SR_Authenticate(M24SR_AccessTypes type, uint8_t* password);
bool M24SR_ChangePassword(M24SR_AccessTypes type, uint8_t* password);
bool M24SR_SetNDEFProtection(M24SR_SetNDEFProtectionActions action);
bool M24SR_SetI2CProtection(bool protect);

//GPO control
bool M24SR_SetGPOMode(M24SR_GPO_Modes mode);
bool M24SR_SetGPO(bool high);
bool M24SR_SendInterrupt(void);
bool M24SR_GPOIsActive(void);
```

```
//RF enabled control
void M24SR_SetRFEnabled(bool enabled);

//NDEF file operations
bool M24SR_GetNDEFFileLength(uint16_t* length);
bool M24SR_SetNDEFFileLength(uint16_t length);
bool M24SR_ReadFromNDEFFileContent(uint8_t* storage, uint16_t offset, uint16_t
    length);
bool M24SR_ReadWholeNDEFFile(uint8_t* storage, uint16_t* length);
bool M24SR_WriteToNDEFFileContent(uint8_t* data, uint16_t offset, uint16_t
    length);
```

Výpis 7: Veřejné prototypy a deklarace knihovny M24SR

6.1.2.1 Relace Jedná se o funkce pro zahájení a ukončení relace (aktivního spojení) mezi MCU a čipem M24SR.

6.1.2.1.1 Ready Nastaví I^2C správnou adresu cílového zařízení a požádá M24SR o zahájení relace. Parametr *forced* určuje, zdali bude požadavek vynucen. V případě nevynucení čip může odmítnout, pokud je anténa obsazena telefonem. V případě vynucení bude anténa spolehlivě odříznuta a k zahájení relace určitě dojde. Návratová hodnota funkce říká, zdali byla relace navázána.

6.1.2.1.2 EndI2CSession Ukončí relaci s čipem M24SR a uvolní tím jeho anténu pro přístup od telefonu.

6.1.2.2 Odezva Slouží k získání odpovědi od M24SR.

6.1.2.2.1 GetResponse Získá od čipu odpověď týkající se výsledku provedení naposledy zadaného příkazu v současné relaci. Čip odpovídá dvěma bajty, a pokud je první roven 144 a druhý 0, znamená to úspěšné provedení, jakákoliv jiná hodnota představuje selhání. Funkce tedy vrací výsledek porovnání se zmíněnými hodnotami.

6.1.2.3 Zabezpečení Skupina funkcí zajišťuje autorizaci k chráněnému přístupu, zapínání a vypínání ochran, a změnu hesel.

6.1.2.3.1 Authenticate Provede autorizaci pro získání příslušného oprávnění. Vstupní argument *type* je enumerátorem dávajícím na výběr mezi čtením či zápisem do NDEF souboru, a nebo přístupem přes I^2C . Z mikrokontroléru má největší smysl užít volbu třetí, neboť je velmi privilegovaná. V tomto projektu je používána pouze tato volba. Jako argument druhý je *password*, což je přístupové heslo. Autorizace je nezbytná pro použití všech 3 dalších funkcí této sekce. Jejich operace se povedou jen v případě, že autorizace bude zahrnovat příslušné oprávnění. Právě autorizace k I^2C přístupu obsahuje všechna existující oprávnění.

6.1.2.3.2 ChangePassword Vstupní argumenty jsou totožné jako u předešlé funkce. Pro zvolený druh přístupu změní přístupové heslo.

6.1.2.3.3 SetNDEFProtection Funkce se vztahuje k zabezpečení přístupu k NDEF souboru. Argument *action* je enumerátorem pro výběr akce, což může být některá z kombinací nastavení či zrušení zabezpečení zápisu či čtení.

6.1.2.3.4 SetI2CProtection Funkce zavádí či odstraňuje zabezpečení pro přístup přes I^2C . V tomto projektu byla ochrana pomocí tohoto příkazu vypnuta. Samozřejmě funkce nemá žádnou obdobu v aplikaci pro telefon.

6.1.2.4 Řízení GPO Skupina funkcí pro kontrolu nad univerzálním pinem GPO.

6.1.2.4.1 SetGPOMode Funkce nastavuje režim práce GPO pinu. Argument *mode* je enumerátor s režimy.

Bohužel praxe ukázala, že většina režimů nefunguje nebo funguje špatně. Polovičně funguje režim pro manuální ovládání pinu, protože to lze provést jen ze strany MCU a ne ze strany telefonu (což musí být chyba výrobce). Jediný spolehlivý režim je *RFBusy*, kdy GPO změní logickou úroveň, když rozpozná přítomnost telefonu u antény (telefon totiž dodává výkon) a setrvává až do jeho opuštění. V tomto projektu je proto tento režim užit a osvědčil se.

6.1.2.4.2 SetGPO Funkce sloužící k manuálnímu nastavení logické úrovně na pinu GPO. Lze použít pouze v pokud je nastaven manuální režim, v opačném případě selže.

6.1.2.4.3 SendInterrupt Funkce slouží k vyslání pulzu (jehož délku stanoví sám čip podle okolností) na pin GPO. Lze použít jen pokud je nastaven režim přerušování, v opačném případě selže. Bohužel praxe ukazuje, že selže vždy.

6.1.2.4.4 GPOIsActive Dotazovací funkce, která vrátí boolean, zdali je pin GPO aktivní (tzn. v nízké úrovni). V použitém režimu *RFBusy* to znamená, že je telefon stále přítomen.

6.1.2.5 Blokace antény Slouží k ovládání funkčnosti antény.

6.1.2.5.1 SetRFEnabled Funkce povoluje nebo zakazuje aktivitu antény. Nejedná se o příkaz, charakter je fyzický, neboť čip má přímo digitální vstup pro tento účel. Funkce je proto realizována komponentou BitIO popsanou již dříve. Pokud je anténa zablokována, telefon vůbec nepozná, že je v blízkosti NFC zařízení.

6.1.2.6 Práce s NDEF souborem Skupina nejdůležitějších funkcí celé knihovny pro samotný přístup do paměti čipu (NDEF soubor je užitnou části paměti).

6.1.2.6.1 GetNDEFFileLength Funkce slouží k zjištění délky obsazené části souboru. Argument *length* je pointer, přes který se vrátí počet obsazených bajtů nepočítaje 2 úvodní, které informaci o délce obsahují.

6.1.2.6.2 SetNDEFFileLength Funkce slouží k nastavení délky obsazen části souboru. Argument *length* je délka, která bude rozložena s pomocí funkce z knihovny DataConversions na dva bajty a zapsána právě do příslušných dvou bajtů na začátku NDEF souboru. Je důležité, aby délka byla vždy správně stanovena, neboť čip má kontrolu přístupu k paměti.

6.1.2.6.3 ReadFromNDEFFileContent Funkce čte užitečnou část NDEF souboru v rozsahu zadaném argumenty. První argument je pointer na úložiště, druhý je index bajtu, od kterého se má začít číst, a třetí je počet bajtů, které mají být přečteny. Funkce nijak nehlídá správnost vstupních hodnot a selže, pokud hodnoty nebudou odpovídat možnostem. Čtení se provádí cyklicky přes buffer o velikosti 64 B. Velikost je stanovena konstantou, kterou lze snadno přepsat. V každém cyklu je kontrolována odpověď čipu, takže dojde k ukončení čtení co nejdříve po případném selhání.

6.1.2.6.4 ReadWholeNDEFFile Funkce je velmi podobná předešlé, jen s tím rozdílem, že NDEF soubor přečte celý od začátku do konce (vyjma velikostních bajtů). Její druhý argument je pointer, přes který se vrátí počet skutečně přečtených bajtů.

6.1.2.6.5 WriteToNDEFFileContent Funkce je duální k funkci *ReadFromNDEFFileContent*, jen provádí opačnou operaci, a to zápis do NDEF souboru. Opět platí, že funkce neuhlídá správné hodnoty a může v průběhu selhat. A také se zápis provádí přes buffer, jehož

velikost určuje konstanta. Praxe však ukázala, že buffer musí být jednobajtový, jinak se v náhodných místech paměti nepodaří zápis a v jiných ano. Je to prazvláštní jev a u velkého výrobce, jako je ST Microelectronics, dosti nečekaný.

6.1.2.7 Celkové užití Knihovna si žádá dodržet určitá pravidla v pořadí volání funkcí a také z vlastností čipu vycházejí omezení. Vždy před jakoukoliv prací s čipem přes sběrnici je nutné začít zavoláním funkce *Ready*, aby se otevřela relace. Při tom je nutné zvážit, zdali je žádoucí ji vynutit, nebo to udělat umírněně. V druhém případě je však zapotřebí všímat si návratové hodnoty, která říká, zdali byla relace zahájena, a při neúspěchu nepokračovat. Naopak po ukončení práce je zapotřebí zavolat funkci *EndI2CSession*, aby se uzavřela relace a anténa se stala opět viditelnou pro telefon. Anténa je totiž po dobu relace blokována a konec relace nenastává samovolně.

Jak si lze všimnout, většina funkcí má návratovou hodnotu bool, indikující, zdali se operaace povedla, avšak vypovídací hodnota není příliš velká. Kontroluje se totiž pouze, zdali nedošlo k chybě na sběrnici I^2C . Pokud je nutné opravdu znát výsledek operace, nezbývá než po jejím provedení zavolat funkci *GetResponse*, která se již opravdu ptá přímo čipu. Bylo by však časově neefektivní používat ji příliš často. Je dobrá praxe ověřit si pomocí zmíněné funkce zahájení relace.

Pro jakoukoliv změnu zabezpečení je nutné nejprve provést autorizaci. Z MCU je výhodné používat zvýšená privilegia pro I^2C přístup. Zapomenutí hesel k NDEF souboru nepředstavuje problém, avšak ztráta I^2C hesla je neřešitelný problém. Pro změnu režimu GPO je také zapotřebí autorizace pro I^2C přístup, pokud je tato ochrana aktivní. Taktéž pokud je NDEF soubor zabezpečen, je zapotřebí se příslušně autorizovat.

Při práci s NDEF souborem je nutno hlídat si velikosti v paměti. NDEF si drží informaci o velikosti obsazené části a čip trvá na jejím respektování. Nelze zapisovat dále, než obsazená část sahá. Před zápisem je nutné ji upravit. Taktéž nelze číst přes limit. Proto má knihovna funkce *GetNDEFFileLength* a *SetNDEFFileLength* a její uživatel se bez nich neobejde. Dlužno na konec zmínit, že zápis je mnohem časové náročnější operace, než čtení. Přepsání celé paměti v maximální možné délce je otázkou možná 10 s, přečtení by trvalo asi čtvrtinu času.

6.1.3 Celkový měřicí program

Na závěr vývoje programu pro KL25Z byly spojeny části předešle popsané v jeden funkční celek. To jest měřicí program, využívající M24SR jako úložiště dat a zároveň jako výměník, přes který telefon zadává příkazy týkající se měření, MCU je vyhodnocuje a odpovídá přepsáním hodnot v paměti. Pro předávání informací byla vyhrazena skupina 13 bajtů na začátku NDEF souboru, což je znázorněno na obrázku 9.

Rozložení systémové části paměti je takovéto:

- Byte 0 může nabývat hodnoty 0 nebo 1, slouží jako boolovská informace.
- Byte 1 obsahuje míru nabití baterie v procentech, tudíž jeho rozsah je 0–100.
- Byte 2 obsahuje číslo se stavem měření z enumerátoru *MeasState*. Využívá se zde skvělé vlastnosti jazyka C, který enumerátor bere jako soubor pojmenovaných čísel a umožňuje explicitní konverzi oběma směry, což zakládá nejelegantnější způsob ukládání enumeračních proměnných, jaký lze vůbec vymyslet.
- Byte 3 obsahuje číslo s typem měření z enumerátoru *MeasType*.
- Bajty 4–8 obsahují datum začátku měření v „lidském“ formátu. Rok je ponížen o 2 milénia, díky čemuž mu postačí jeden bajt (v roce 2256 však dojde k selhání).
- Bajty 9–10 obsahují společně periodu měření v sekundách. Bajty 11-12 obsahuj společně počet vzorků, který má být nasbírán.

Obr. 9: Diagram systémových bajtů v NDEF souboru

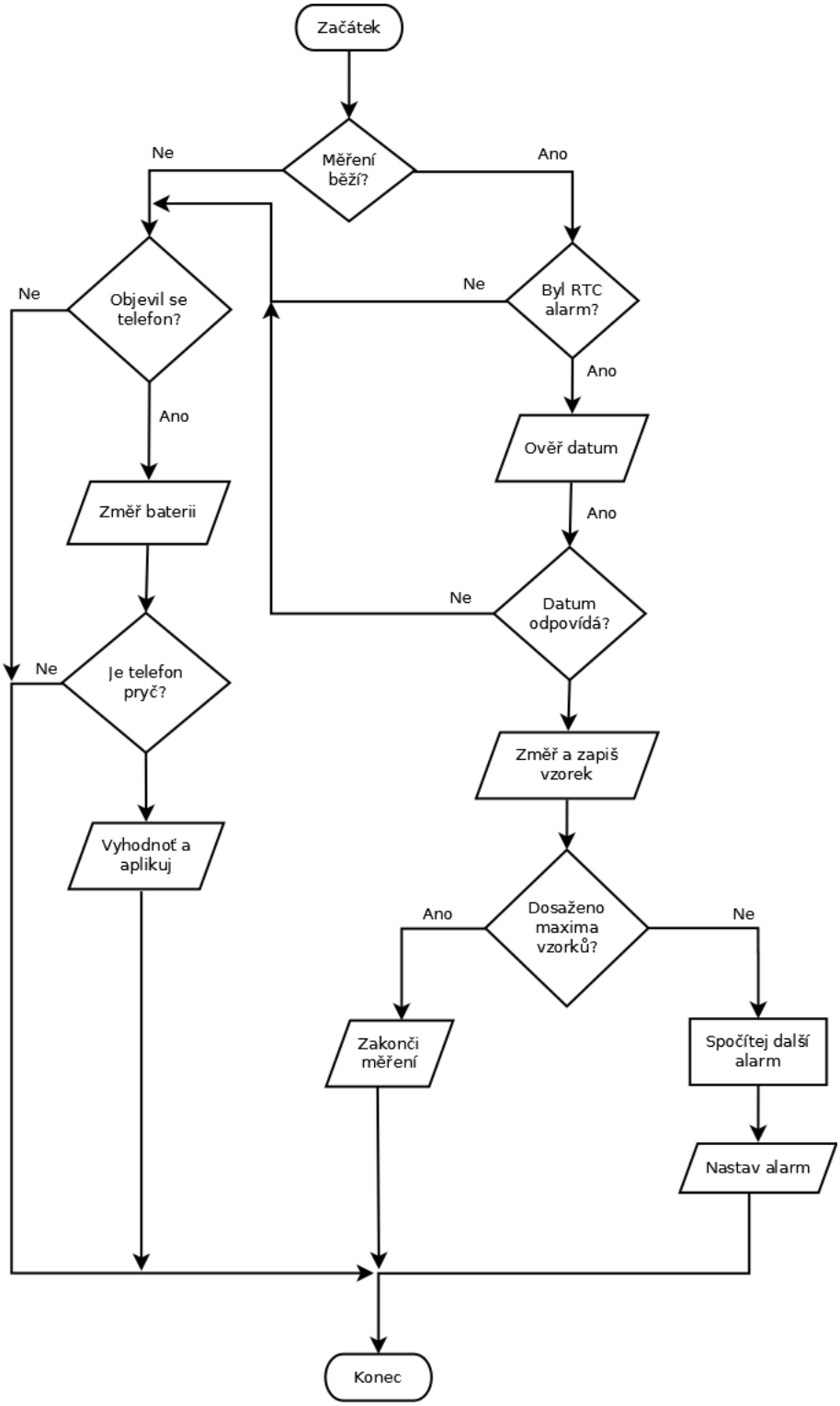
```
typedef enum
{
   MeasState_None,
   MeasState_MCUEvaluationPending,
   MeasState_WaitingForStartTime,
   MeasState_Running,
   MeasState_FinishedSuccessfullyAuto,
   MeasState_FinishedSuccessfullyManual,
   MeasState_Failed,
   MeasState_FailedLowBattery
} MeasSystem_MeasState;

typedef enum
{
   MeasType_None,
   MeasType_Voltage
```

```
} MeasSystem_MeasType;

void MeasSystem_GPOHandler(int current_time);
void MeasSystem_Periodic(int current_time);
void MeasSystem_RTCRoutine(void);
```

Výpis 8: Veřejné deklarace knihovny MeasSystem



Obr. 10: Vývojový diagram systému na KL25Z

Co do programové struktury vznikla knihovna MeasSystem, která zaobaluje celou funkčnost. Jejími vstupními body ze základního programu jsou 3 funkce, které jejichž prototypy jsou ve

výpisu 8.

Funkce *GPOHandler* je volána z komponenty ExtInt navázané na GPO pin. Ten je v režimu *RFBusy*, takže se překlápí do nízké úrovni při přítomnosti telefonu. Funkce tedy představuje rutinu a proto pouze nastavuje příznak *evalPending* znamenající nutnost provést vyhodnocení a ukládá do proměnné momentální počet sekund, aby mohlo být vyhodnocení zpožděno.

Funkce *RTCRoutine* je volána z komponenty TimeDate, což jsou hodiny reálného času. Je to rutina při dosažení nastaveného času alarmu. Ta pouze nastavuje příznak *RTCPending*, aby mohl být obsloužen.

Funkce *Periodic*, jak název napovídá, je volána periodicky, a to z hlavní smyčky a za pomocí dříve popsaného časování, kde perioda je přibližně 10 ms. V ní jsou vykonávány všechny operace, v závislosti na příznacích nastavených 2 výše popsanými rutinami. Vývojový diagram této funkce popisující funkčnost systému je zobrazen na obrázku 10. Za každým rozhodovacím blokem se skrývá porovnávání proměnných a za akčními bloky volání interních funkcí knihovny. Zde jsou popsány blíže postupy při posunu diagramem.

Ověření běhu měření se provede pouhým porovnáním boolovského příznaku. To, že nastal alarm hodin reálného času indikuje příznak, který byl zmíněn u jejich rutiny. Ověření data se provádí získáním aktuálního data od hodin reálného času a porovnáním s datem následujícího měření, obě struktury se po položce porovnají.

Měření vzorku probíhá zavoláním funkce *measure*, která na základě zvoleného druhu měření v bloku switch provede získání vzorku a uloží jej do NDEF souboru s pomocí knihovny M24SR po rozložení na bajty (díky knihovně DataConversions), samozřejmě navýší jeho velikost odpovídajícím způsobem. Pokud se jednalo o poslední vzorek, protože byl dosažen cílový počet vzorků, je měření zastaveno a do NDEF souboru je jako stav měření zapsáno úspěšné automatické dokončení.

Pokud se nejednalo o poslední vzorek, je k datum inkrementováno o periodu v sekundách ve funkci *calculateNextAlarm*, která cyklicky sekundu po sekundě přičítá k datu a kontroluje náležitosti, jako je počet dnů v měsíci i přestupné roky. Alarm je pak nastaven komponentě hodin reálného času.

Přítomnost telefonu se zjistí podle dříve zmíněného příznaku z rutiny GPO. Změření baterie je prostým změřením napětí pomocí A/D převodníku na pinu připojeném ke kladné svorce baterie. Ověření, zdali telefon je již pryč, spočívá v kontrole, zdali je pin GPO v normální úrovni a zdali už od posledního přerušení uplynul čas daný konstantou, ideální jsou 4 s.

Vyhodnocení se provede zavoláním funkce *evaluate*, ve které se kontroluje průběžně několik věcí. Za prvé se zjistí, jestli NDEF soubor není prázdný. Pokud je prázdný a zároveň je načasováno měření, je toto měření zrušeno. Pokud není prázdný, je přečtena systémová část. Na začátek je zkontrolován první bajt, značící nutnost vyhodnocení, a pokud je nulový, funkce skončí. Poté

se zkontroluje, jestli telefon nepřenastavil stav běžícího měření na manuální zastavení. Pokud ano, měření se ukončí. Jako další se ověří, zdali stav měření není „čekající na vyhodnocení“. Pokud je, skončí se. Do poslední sekce funkce lze dojít jen v případě, že telefon vložil nové měření. V této situaci se jen ze systémových bajtů poskládají informace o měření (začátek, druh, počet vzorků, perioda), nastaví se stav „čekající na začátek“ a načasuje se alarm prvního měření.

Při měření je úmyslně započetí relace s čipem M24SR vynuceno, aby nedošlo k selhání. Pokud by došlo k selhání při získávání vzorku, je celé měření označeno za neúspěšné a zastavuje se. Uživatel pak bude moct přistoupit k datům, avšak budou necelá. Také se kontroluje stav baterie a pokud klesne pod mez nastavenou konstantou, dojde k ukončení měření a stav se zapíše jako „neúspěšně ukončeno kvůli baterii“. Je totiž velmi žádoucí, aby nedošlo k jejímu úplnému vybití. To by poté mikrokontrolér ztratil napájení a resetovala by se informace o čase.

Po skončení oběhu smyčky přejde mikroprocesor do úsporného režimu *WAIT*, který představuje zastavení jádra do dalšího přerušení. Tudíž do doby, než dojde k detekci telefonu nebo alarmu na hodinách reálného času. A tedy v době mezi akčními okamžiky měření má mikroprocesor silně sníženou spotřebu a vyhovuje nízkoodběrovosti, protože tyto doby jsou u pomalého měření značně dlouhé a ve výsledku je v chodu zlomek času z celkového trvání. Způsob přechodu do tohoto režimu je velmi jednoduchý, neboť automaticky přidaná komponenta s názvem *MKL25Z128VLK4* obsahuje funkci pro tento účel. Způsob použití je znázorněn ve výpisu 9. Volaná funkce má jako první argument enumerační položku z komponenty a ostatní 2 argumenty podle dokumentace ignoruje, takže jsou tam nulové konstanty, protože na tom nezáleží.

```
Cpu_SetOperationMode(DOM_WAIT, NULL, NULL);
```

Výpis 9: Provedení přechodu do úsporného režimu [7]

6.2 Uživatelská aplikace pro Android

Pro uživatelské ovládání celého měřicího systému byla vyvinuta aplikace pro mobilní telefony s operačním systémem Android. Vývoj byl proveden za pomoci SW Android Studio z důvodů popsaných v návrhu. Pro telefon byl nejprve vytvořen funkční základ aplikace, který je znovupoužitelný. Zahrnuje řadu potřebných algoritmů a poskytuje prostředí pro tvorbu konkrétní aplikace. Poté byla s pomocí tříd pro NFC vytvořena třída konkrétně pro čip M24SR. S jejím použitím byla vyvinuta následně pro potřeby tohoto projektu specializovaná aplikace.

6.2.1 Obecný základ aplikace

Zmíněný základ obsahuje řadu tříd, které usnadňují běžné úkony, které je zapotřebí opakovaně provádět. Část zabývající se problémy netýkajícími se GUI řeší převody mezi formáty dat, zís-

kávání a převod časových informací, operace v souborovém systému, ověřování formátu vstupů, vzdálené webové požadavky, užívání permanentního úložiště hodnot, a konečně základní přístup přes NFC. Ostatní třídy se zabývají GUI a jejich náplní je práce s Activity, zobrazování dialogových oken různého druhu a podobná usnadnění.

6.2.1.1 Převod formátu dat

Vznikla ryze statická třída *DataConversions*, která je částečně analogií stejnojmenné knihovny v programu mikrokontroléru, protože i zde je nutné řešit rozkládání a skládání bajtů, neboť se přímo přistupuje k čipu M24SR. Avšak díky vysokému jazyku Java je tato třída obecnější a pokryje více případů. Ze stejného důvodu však musí být přizpůsobená na velký nešvar tohoto jazyka, a to ten, že neobsahuje bezznaménkové datové typy (slovo „unsigned“ zde nic neznamená), což je překážka při práci s daty na nízké úrovni, a od tvůrců tohoto jazyka je to nepochopitelné, až přímo trestuhodné, pochybení. Výčet veřejných metod této třídy je ve výpisu 10, speciální komentář není nutný.

```
public static int ByteToInt(byte Byte)
public static int[] BytesToInts(byte[] bytes)

public static byte[] IntegerToBytes(long number, int byteCount)
public static byte[] IntegerToBytes(long number)
public static long BytesToInteger(byte... bytes)
public static long BytesToInteger(byte... bytes)

public static byte[] ShortsToBytes(short[] shorts)
public static short[] BytesToShorts(byte[] bytes)
```

Výpis 10: Veřejné metody třídy DataConversions

6.2.1.2 Časové informace

Už jen vzhledem k povaze měřicího systému vzniká potřeba pracovat s časovými údaji, formátovat časové údaje do textových řetězců a parsovat je nazpátek. Android umožňuje formát zadávat formátovacími řetězci a nebo formát získat na základě regionálního nastavení operačního systému. Oba způsoby implementuje ryze statická třída *DateTimeInformation*, jejíž veřejné členy jsou vypsány ve výpisu 11.

```
public static final String DATE_USA = "MM/dd/yyyy";
public static final String TIME_USA_NOSEC = "hh:mm aa";
public static final String TIME_USA_WITHSEC = "hh:mm:ss aa";
public static final String DATE_UK = "dd/MM/yyyy";
public static final String DATE_CZECH = "dd.MM.yyyy";
```

```
public static final String TIME_UNIV_NOSEC = "HH:mm";
public static final String TIME_UNIV_WITHSEC = "HH:mm:ss";
public static Date StringToDate(String dateFormat, String dateString)
public static String DateToString(String dateFormat, Date date)

public static final int LOCALE_DATE_FULLYEAR = DateFormat.MEDIUM;
public static final int LOCALE_DATE_LESSYEAR = DateFormat.SHORT;
public static final int LOCALE_TIME_WITHSECONDS = DateFormat.MEDIUM;
public static final int LOCALE_TIME_NOSECONDS = DateFormat.SHORT;
public static final int LOCALE_TIME_NONE = -1;
@IntDef({LOCALE_DATE_FULLYEAR, LOCALE_DATE_LESSYEAR})
@interface LocaleDateLengths{}
@IntDef({LOCALE_TIME_NOSECONDS, LOCALE_TIME_WITHSECONDS, LOCALE_TIME_NONE})
@interface LocaleTimeLengths{}
public static String DateToLocaleString(Date date, @LocaleDateLengths int
    dateLength, @LocaleTimeLengths int timeLength)
```

Výpis 11: Přehled veřejných členů třídy DateTimeInformation

První 2 metody slouží pro snadné formátování a parsování, lze zadat přímo formátovací řetězec a nebo zvolit nějaký předdefinovaný zadáním cesty jedné z konstant výše. Metoda *DateToLocaleString* slouží k formátování data v závislosti na regionálním nastavení. Dává na výběr z několika možných formátů, protože vstupní argumenty jsou limitovány pomocí argumentních interface.

6.2.1.3 Souborové operace Protože vnějším výstupem měřicího procesu je CSV soubor, plyne nutnost umět do souborového systému uložit soubor a umět jej opět smazat. Při této příležitosti byly implementovány i další užitečné postupy, které však nejsou aplikovány, jako je načtení souboru, nalistování souborů v adresáři a další. Tyto potřeby pokrývá ryze statická třída *FileOperations*, jejíž výčet metod je ve výpisu 12.

```
public static String[] GetBasenameAndExtension(String filename)

public static boolean SavePublicFile(String directory, String filename, boolean
     canOverwrite, byte[] content)
public static boolean SavePrivateFile(Context context, String filename, byte[]
    content)

public static byte[] LoadPublicFile(String directory, String filename)
```

```
public static byte[] LoadPrivateFile(Context context, String filename)

public static File[] ListFilesInPublicDirectory(String directory)

public static int DeleteFiles(File... files)
public static int DeletePublicDirectory(String directory)
public static int DeleteAllPrivateFiles(Context context)
```

Výpis 12: Přehled veřejných metod třídy FileOperations

V aplikaci jsou reálné užity jen některé metody, které nyní budou zmíněny:

- Metoda *GetBasenameAndExtension* rozdělí název souboru na jméno a příponu a vrátí v podobě dvouprvkového pole.
- Metoda *SavePublicFile* uloží soubor ve veřejném úložišti v daném adresáři (pokud neexistuje, vytvoří jej). Argument canOverwrite znamená, zdali může být soubor přepsán. Pokud je nepravdivý, zkontroluje se existence souboru a namísto přepsání dojde k očíslování.
- Metoda *DeletePublicDirectory* smaže adresář ve veřejném úložišti. Metoda používá rekurzivní algoritmus, díky čemuž spolehlivě smaže i složku s více podsložkami a soubory, aniž by větvení bylo omezující.

6.2.1.4 Kontrola formátu vstupů

V této aplikaci jsou 2 údaje, které uživatel zadává pomocí klávesnice, a to je heslo pro zabezpečení a emailová adresa pro zaslání CSV souboru. V obou případech je žádoucí ověřit validitu vstupu. Proto byla vytvořena ryze statická třída *Formatting*. Přehled jejích metod je ve výpisu 13.

```
public static boolean CheckEmailValidity(String email)
public static boolean CheckPasswordValidity(String password, int min, int max)
```

Výpis 13: Přehled veřejných metod třídy Formatting

Metoda *CheckEmailValidity* ověřuje, zdali zadaná adresa vyhovuje obecným předpokladům pro emailovou adresu. Ta musí určitě obsahovat zavináč a sestávat jen z malých písmen anglické abecedy a povolených oddělovačů. Adresa musí končit doménou nejvyššího řádu zleva oddělenou tečkou.

Metoda *CheckPasswordValidity* ověřuje, zdali heslo obsahuje pouze malá a velká písmena anglické abecedy a číslice. Ověřuje také rozsah délky zadaný dvěma vstupními argumenty.

6.2.1.5 Síťové nástroje

Protože aplikace nabízí možnost odeslat CSV soubor na emailovou adresu, musí k tomu být adekvátní způsob. Bylo zvoleno zprostředkování PHP skriptem na webovém serveru a uložení do tamního úložiště. Pro tento způsob je nutné poslat na web požadavek emulující vstup, jaký by odpovídal webovému prohlížeči. Tuto funkcionalitu obstarává ryze statická třída *NetTools*, jejíž 2 metody jsou ve výpisu 14.

```
public static boolean PingGoogleDNS()
public static String WebRequest(String url, String POSTparams)
```

Výpis 14: Přehled veřejných metod třídy Formatting

Zmíněné odeslání souboru je záležitost metody *WebRequest*. Ta má 2 argumenty, první je URL adresa, druhý jsou parametry, které se předají metodou POST. Podle pravidel HTTP požadavků je zapotřebí používat stejné názvy parametrů, jako jsou proměnné v PHP skriptu. Návratovou hodnotou metody je obdržená odpověď, která se odvíjí od podoby skriptu.

Velmi užitečná je metoda *PingGoogleDNS*, která slouží pro ověření správného připojení k síti. Systém Android má zvláštnost, že pro zjištění připojení má jiné oprávnění, než pro používání připojení, a stejně dokáže jen říct, že je připojena WiFi, avšak jestli je router připojen k nějaké WAN síti už nezjišťuje. Proto je nejspolehlivějším způsobem zkusit se reálně někam připojit. A samozřejmě nejúspornějším řešením je operace ping, která se jen zeptá za odpověď. Je však zapotřebí zeptat se spolehlivého hostitele, a žádný nemůže být tak spolehlivý, jako největší DNS server na světě na IP adrese 8.8.8.8 patřící společnosti Google. Tato metoda použije interní terminál OS Android a vrátí boolovskou informaci, zdali byla úspěšná. Volání a vyhodnocení této metody se dá označit za prerekvizitu pro webový požadavek.

6.2.1.6 Permanentní úložiště hodnot

Pro uložení proměnných, které nebudou ztraceny po ukončení aplikace, nabízí jako nejsnazší cestu OS Android spravované permanentní úložiště pokryté třídou *SharedPreferences*. Systém vložené hodnoty ukládá sám do systémového utajeného adresáře nejspíše v podobě XML souboru. Je to efektivní nástroj především pro ukládání zapamatovaných voleb uživatele. Systém zřejmě vyhrazuje aplikaci vlastní adresář a třída má v konstruktoru jako argument zadání názvu množiny hodnot, což značí na uložení do jednotlivých souborů. Každá hodnota má pak své jméno. Uplatňuje se princip zrcadlení, kdy aplikace dostane od systému kopii souboru s hodnotami, a všechny změny provádí právě v této kopii a teprve na konci všech žádaných úprav předá systému kopii na zpět a požádá ho o přepsání původního souboru touto kopií. Z toho plyne, že je možné změny kdykoliv před potvrzením odřeknout.

Třída *SharedPreferences* způsobem přístupu dokonalá a proto byla vytvořena vlastní třída *SharedPreferencesManager* s přirozenějším přístupem. Přehled veřejných členů třídy je ve výpisu 15.

```
public enum Types
{
   String,
   Boolean,
   Integer,
   Long,
   Float
}

public SharedPreferencesManager(Context context, String filename)

public Object GetValue(Types type, String name)
public void SetValue(Types type, String name, Object value, boolean commit)
public void RemoveValue(String name, boolean commit)
public void Clear()
public void CommitChanges()
```

Výpis 15: Přehled veřejných členů třídy SharedPreferencesManager

Význam a použití těchto metod je následující:

- Konstruktor si jako argument žádá referenci na Activity a název souboru hodnot
- Metoda *GetValue* požaduje výběr typu z enumerátoru *Types* a název hodnoty. Hodnotu vrátí v podobě Object, který je pak zapotřebí explicitně přetypovat.
- Metoda nastaví danou hodnotu v úložišti a *SetValue* požaduje typ, název, cílenou hodnotu a posledním argumentem je boolovský příznak, zdali má být změna ihned aplikována (nebo budou naopak následovat další).
- Metoda *RemoveValue* smaže hodnotu z úložiště a požaduje její jméno a také má příznak pro okamžité vykonání.
- Metoda *Clear* kompletně promaže celý soubor hodnot a ihned změnu aplikuje.
- Metoda *CommitChanges* slouží pro případ, kdy je postupně prováděno více změn, neboť všechny neuložené změny aplikuje.

6.2.1.7 Prvky uživatelského rozhraní Kvůli možnosti předávat třídám spravujícím části grafického uživatelského rozhraní funkční kusy kódu je, v vhledem k filosofii jazyka Java, nutno použít interface. Již existující intefrace Runnable by byl vhodným kandidátem, avšak nepodporuje předání argumentů. Proto byl vytvořen vlastní interface *RunnableWithParams*, který je zobrazen ve výpisu 16.

```
public interface RunnableWithParams
{
   boolean run(Object... param);
}
```

Výpis 16: Interface RunnableWithParams

Jedním ze základních opakovaně používaných prvků je dialog překrývající pracovní plochu zabraňující ovládat prvku GUI. Jsou vhodné pro zobrazení nějaké výzvy, specifický textový vstup či jako potvrzovací a čekací dialogy. Pro tyto účely byla vytvořena bohatá třída *AlertDialogsManager*, která poskytuje nástroje pro obecné užívání těchto dialogů. Přehled veřejných členů této třídy je ve výpisu 17.

```
public enum SelectionTypes
{
   Simple,
   SingleChoice,
   MultiChoice
}

public AlertDialogsManager(Activity activity)

public void DismissDialog()
public void ShowUnsavedChangesDialog(String message, final RunnableWithParams
    ... runBeforeExit)
public void ShowConfirmationDialog(String message, final RunnableWithParams
    onConfirm, final RunnableWithParams onCancel)
public void ShowSimpleMessagebox(String title, String message)
public void ShowSelectionMessagebox(final SelectionTypes type, String title,
    String[] items, @Nullable Object selectedItems, final RunnableWithParams
    onPositive)

public void ShowWaitingSpinner(String message, @Nullable final
    RunnableWithParams onCancel)
public void ChangeSpinnerMessage(String message)
```

Výpis 17: Přehled veřejných členů třídy AlertDialogsManager

Protože je tato třída využívána hodně, budou nyní popsány metody:

- Konstruktor požaduje referenci na Activity
- Metoda *DismissDialog* slouží ke skrytí dialogu, který je právě zobrazen.
- Metoda *ShowUnsavedChangesDialog* slouží k zobrazení potvrzovací výzvy, když uživatel stiskne tlačítko zpět v situaci, kdy ukončení Activity způsobí ztrátu neuložených dat. Jeho první argument je zobrazený text a druhý je volitelný argument, který s využitím výše popsaného interface umožňuje předat kód, který bude vykonán ještě před ukončením aktivity. Uživatel uvidí tlačítko pro zrušení své akce a pro potvrzení, kde potvrzení znamená právě ono skončení aktivity.
- Metoda *ShowConfirmationDialog* je obecnějším případem předchozí metody. Zobrazí potvrzovací dialog s vloženým textem a tlačítko pro potvrzení a storno. Podle toho, co uživatel zmáčkne, vykoná kód v jednou z dvou posledních argumentů. Slouží k použití ve chvílích, kdy uživatel vyvolal nějakou akci a je vhodné zdůraznit mu závažnost a požádat ho o definitivní rozhodnutí, čímž vina přechází na něj.
- Metoda *ShowSimpleMessagebox* je velmi jednoduchá na použití a zobrazí nejzákladnější dialog, který má v záhlaví titulek, uvnitř text a jediné tlačítko OK.
- Metoda *ShowSelectionMessagebox* slouží k zobrazení dialogu se seznamem položek. První argument je enumerační a určuje, jaký režim bude užit. Naprosto zásadním způsobem rozděluje funkčnost na 3 možnosti:
  - Režim *Simple* znamená, že položky se budou chovat jako veliké tlačítko a po stisknutí některé se dialog sám zavře. Předposlední argument v tomto režimu nemá žádný význam.
  - Režim *SingleChoice* znamená, že položky budou mít radio tlačítko a vybrána může být v kteroukoliv chvíli právě jedna. Předposlední argument pak musí být index na začátku vybrané položky.
  - Režim *MultiChoice* znamená, že položky budou mít checkbox tlačítko a vybrán může být libovolný počet. Předposlední argument musí být pole boolean značící, které položky mají být na začátku označeny.
- Metoda *ShoWaitingSpinner* zobrazí dialog s točícím se kolečkem a zadanou zprávou, který slouží jako výzva při čekání s neurčitelnou délkou. Druhý argument volitelně umožňuje provést akci při zrušení dialogu tlačítkem zpět.
- Metoda *ChangeSpinnerMessage* slouží ke změně zprávy v čekacím dialogu, když je zrovna zobrazen.

Základním stavebním celkem aplikace je Activity, která představuje jednu celistvou obrazovku, přičemž aktivní může být vždy jen jedna v jednu chvíli. Velmi často je pro vyobrazování

hlášek a upozornění užívána textová bublina Toast i vyjížděcí panel Snackbar. Syntaxe pro jejich použití by se dala elegantně redukovat. Taktéž je dobrý nápad nějakou standardně uplatňovat doposud popsané třídy. Proto byla vytvořena abstraktní třída *ActivitySmart* jakožto potomek třídy *AppCompatActivity*, a slouží jako podklad. Není nutné přidávat výpis, stačí zmínit, že řeší snazší zobrazení Toast i Snackbar, obsahuje v globální proměnných instance některých vlastních tříd a má některé užitečné metody, např. pro skrytí klávesnice apod.

6.2.2 Implementace NFC

V první řadě byly implementovány nejobecnější algoritmy pro práci s NFC komponentou a poté s jejich využitím vytvořeny konkrétní postupy pro čip M24SR.

6.2.2.1 Základní přístup Android poměrně dobře zpracované třídy týkající se NFC a podporuje všechny typy. Nejdůležitější třídou je *NfcAdapter*, která přistupuje k HW komponentě bez rozlišení konkrétního typu, je obecná. Zásadní je potom třída *Tag*, která zaštiťuje základní funkčnost tagu jako takového. NFC je svou povahou velmi nepředvídatelný komunikátor a jeho použití je dalo by se říci nárazové. Předpokládá se použití, kdy uživatel provádí nějaké změny či požadavky a aby je aplikoval, přiloží telefon k tagu. Proto informace od NFC mají asynchronní charakter. Android z tohoto důvodu pojímá výskyt tagu v dosahu antény jakožto systémovou událost. Pokud chce aplikace pracovat s tagy, musí konkrétní *Activity* oznámit systému, že umí obsloužit *Intent* s akcí znamenající objevení tagu, a systém pak při události pošle *Intent*, ve kterém bude obsažen *Tag*, který vyhovuje interface *Parcelable*. Z toho plyne že daná *Activity* musí přepsat mateřskou metodu *onNewIntent* a zpracovávat argumenty. Aby *Activity* neblokovala přístup jiných aplikací, musí při pozastavení odregistrovat svou schopnost.

Pro tyto základy byla vytvořena ryze statická třída NFCHandling, jejíž přehled metod je ve výpisu 18.

```
public static boolean IsNFCEnabled(Context context)
public static void SetHandlingAbility(boolean enabled, Activity activity)
public static boolean TagAppeared(Intent intent)
public static Tag GetTag(Intent intent)
public static String GetTechList(Tag tag)
public static String GetNDEFMessage(Intent intent)
```

Výpis 18: Přehled veřejných metod třídy NFCHandling

Použití a význam jednotlivých metod je takovýto:

- Metoda *IsNFCEnabled* získá NFC adaptér a zjistí, zdali je na telefonu zapnuté NFC či nikoliv ěk. Aplikace v tomto projektu nemůže fungovat bez zapnutého NFC, a tak je žádoucí upozornit uživatele.
- Metoda *SetHandlingAbility* registruje v systému Android schopnost zpracovávat odhalení tagu, či ji odregistruje, v závislosti na prvním argumentu.
- Metoda *TagAppeared* z poskytnuté instance *Intent* zjistí a vrátí, zdali obsahuje informaci o tagu. V opačném případě totiž nemá význam.
- Metoda *GetTag* z poskytnuté instance *Intent* získá tag.
- Metoda *GetTechList* zjistí u poskytnutého tagu, kterým specifikacím vyhovuje a sestaví seznam. Specifikace jsou např. NFC-A, NFC-B, IsoDep, NDEF a další.
- Metoda *GetNDEFMessage* z poskytnuté reference na *Intent* načte NDEF zprávu, pokud je k dispozici. NDEF zpráva lze zapsat běžnými NFC aplikacemi jakožto standardizovaný formát. V případě tohoto projektu je nežádoucí a její přítomnost je na závadu.

6.2.2.2 Přístup k M24SR Cílem této fáze bylo jakýmsi způsobem zrcadlit knihovnu z MCU do třídy v Javě. Implementace je odlišná, má klady i zápory. Ve vysokém jazyce se dobře pracuje s poli a dá se zavést mnoho zobecnění, jako např. pro příkazy si vytvořit konstanty ve zvláštní třídě. Na druhou stranu je zde nevýhoda kvůli absenci bezznaménkových datových typů, což působí zbytečně komplikace. Je však skvělé, že NFC adaptér si sám řeší kontrolní algoritmy CRC-16 a není nutné je implementovat, jakožto také při každém příkazu získává od tagu odpověď a to v bezchybné podobě.

Čip M24SR vyhovuje standardu IsoDep, jehož funkčnost řeší stejnojmenná třída. Vyhovuje i dalším standardům, ovšem IsoDep umožňuje funkčnost přesahující klasické NDEF formáty potřebnou pro tento projekt. Před sofistikovaným použitím tedy určitě bude zapotřebí ověřit si, zdali objevený tag vyhovuje této specifikaci.

Výchozí potřebou je získat tag, což už řeší třída *NFCHandling*. Třída *IsoDep* má metodu *get*, která si jako argument bere právě tag a vytvoří instanci pro svůj standard, pokud tag vyhovuje. Pak už je vstupním bodem jediná metoda *transceive*, jako argument bere pole bajtů, které odvysílá k čipu a po přijetí odpovědi vrátí pole bajtů s odpovědí. Tím končí podpora existujících tříd a začíná nutnost implementovat algoritmy konkrétních příkazů.

Před samotnou implementací vznikla pomocná třída *OperationResult*, která slouží k předávání výsledků operací a využívá principu předávání objektů referencí. Díky tomu nemusí mít funkční metody nutně návratový typ boolean. Veřejné členy této třídy jsou ve výpisu 19 bez dalšího komentáře.

```
public enum Results
{
    Success,
    TagLost,
    InvalidCommand,
    UnspecifiedFailure,
    NDEFFileOverflow,
    WrongGPOMode,
    PasswordTooLong,
    PasswordNull,
    PasswordRequired,
    WrongPassword,
    AccessDenied,
    UnexpectedNDEF
}

public void SetResult(Results result)
public Results GetResult()
public void DecideResult(byte[] bytes)
public boolean IsSuccessful()
```

Výpis 19: Přehled veřejných členů třídy OperationResult

Poté už nezbývalo než vytvořit ryze statickou třídu příhodně pojmenovanou *M24SR*, která ovládání čipu zajišťuje. Je to třída velmi robustní a její vyladění nebylo jednoduché. Přehled veřejných členů je ve výpisu 20.

```
public enum AccessType
{
   Read,
   Write
}

private static final int MAX_BYTES = 8190;

public static boolean CheckTagCompatible(Intent intent)

public static int GetNDEFFileContentLength(OperationResult result, Intent
    intent)
```

```
public static void SetNDEFFileContentLength(int length, OperationResult res,
    Intent intent)

public static byte[] ReadFromNDEFFileContent(int offset, int length,
    OperationResult result, Intent intent)
public static byte[] ReadFromNDEFFileContent(OperationResult result, Intent
    intent)

public static void AppendToNDEFFile(int current_length, byte[] data,
    OperationResult result, Intent intent)
public static void InsertIntoNDEFFile(int current_length, byte[] data, int
    offset, OperationResult result, Intent intent)
public static void OverwriteNDEFFile(byte[] data, OperationResult result,
    Intent intent)

public static void AuthenticateNDEFAccess(AccessType type, String password,
    OperationResult result, Intent intent)
public static void AuthenticateNDEFAccess(AccessType type, OperationResult
    result, Intent intent)
public static void SetNDEFProtection(AccessType type, boolean protect,
    OperationResult result, Intent intent)
public static void ChangeNDEFPassword(AccessType type, String password,
    OperationResult result, Intent intent)
```

Výpis 20: Přehled veřejných členů třídy M24SR

Metoda *CheckTagCompatible* vyniká oproti ostatním, jejím úkolem je zkontrolovat, jestli tag vyhovuje specifikaci IsoDep, což zjistí s pomocí načtení listu specifikací a porovnáním. Ostatní metody mají shodný rys v posledním dvou argumentech. Význam *OperationResult* už byl popsán a proč je zapotřebí *Intent* také.

Na začátku používání třídy je zapotřebí autorizovat se metodou *AuthenticateNDEFAccess*, u které se zvolí, zdali se žádá zápis nebo čtení, a předá se heslo. Pokud je přístup nezabezpečený, použije se druhého přetížení, kde heslo nefiguruje. Poté je možno používat neomezeně metody pro druh přístupu, který byl autorizován. Velký problém způsobila neomalená nedomyšlenost od výrobce čipu, kterou bylo vůbec těžké odhalit, a to tu, že povolení k přístupu pro zápis nezahrnuje povolení ke čtení.

Pro zjištění velikosti užité části souboru slouží metoda *GetNDEFFileContentLength* a pro její změnu *SetNDEFFileContentLength*.

Pro čtení slouží metoda *ReadFromNDEFFileContent*, která má 2 přetížení, první čte ze souboru vybraný úsek, druhé ho přečte celý (jeho užitou část).

Pro zápis existují 3 metody. Metoda *OverWriteNDEFFile* přepíše polem bajtů existující data a nataví délku na délku onoho pole. Metoda *AppendToNDEFFile* požaduje informaci o aktuální délce a soubor zvětší a bajty připojí na konec. Metoda *InsertIntoNDEFFile* slouží k vložení pole bajtů kamkoliv dovnitř souboru a je sofistikovaná, protože při neshodě délky pole a délky v souboru se umí rozložit na kombinaci rekurze a metody *AppendToNDEFFile*, neselže tedy na rozdíl od analogické metody na MCU.

Co se týče úpravy zabezpečení, metoda *SetNDEFProtection* nastaví ochranu pro zvolený druh přístupu a metoda *ChangeNDEFPassword* pro něj změní heslo. Samozřejmě je zapotřebí nejprve se autorizovat příslušným způsobem, a to znamená autorizovat se pro zápis. Právo číst pro použití těchto 2 metod nestačí.

Na rozdíl od MCU, kde byla nutnost ukončovat relaci příkazem, se zde ukončuje sama po oddálení antény.

6.2.3 Konkretizace aplikačního základu

Je zapotřebí navázat na dříve popsaný základ aplikace a vytvořit si další stupeň, což je konkretizovaný základ pro ovládání měřicího systému s využitím NFC.

V první řadě vzniká potřeba specializovaných dialogů. Bylo stanoveno, že uživateli nebude umožněno aplikaci používat bez zapnutého NFC, což volá po dialogovém okně, kde bude výzva k zapnutí a při její ignoraci se aplikace vypne. Taktéž když se bude vyžadovat spojení s čipem, nejlepší způsob je, po vzoru ostatních veřejných aplikací, zobrazit dialog s výzvou k přiložení telefonu, která samovolně zmizí, jakmile dojde k dokončení operací. Pro zadání hesla a emailové adresy (nezávisle) při zjištění, že jsou potřebnými údaji, je dobrý nápad užít dialog s políčkem pro textový vstup. Což je případ složitější a žádá si vlastní layout. Protože tyto dialogy jsou opakovaně potřebné, byla pro ně vytvořena třída *AlertDialogsManagerNFC*, která dědí třídu *AlertDialogsManager*, a jejíž veřejné metody jsou zobrazeny ve výpisu 21.

```
public AlertDialogsManagerNFC(Activity activity)

public void ShowNFCDisabledDialog()
public void ShowNFCActionRequirement(final boolean crucial, final
    RunnableWithParams resetAction)
public void ShowEnterPasswordDialog(final RunnableWithParams onEnter, String
    defaultPassw)
public void ShowEnterEmailDialog(final RunnableWithParams onEnter, final
    RunnableWithParams onCancel, String defaultEmail)
```

Výpis 21: Přehled veřejných metod třídy AlertDialogsManagerNFC

Význam metod kromě konstruktoru je ve stručnosti popsán zde:

- Metoda *ShowNFCDisabledDialog* nemá ani žádné argumenty, jen zobrazí varování, že je vypnuté NFC, a že ho uživatel musí zapnout. Dialog má tlačítko OK, které buďto skryje dialog nebo rovnou ukončí Activity, podle toho zdali mezitím uživatel vyhověl výzvě či nikolivěk.
- Metoda *ShowNFCActionRequirement* zobrazí výzvu, aby byl telefon přiložen k anténě čipu. Dialog má 1 tlačítko pro zrušení. První argument je boolovský a značí, jestli se má při zrušení rovnou ukončit aktivita, pokud je pravdivý, druhý argument nemá význam. Když je nepravdivý, druhý argument je akce, který se vykoná při zrušení.
- Metoda *ShowEnterPasswordDialog* zobrazí dialog s textovým polem, ve kterém se znaky nahrazují hvězdičkami, a má tlačítko pro potvrzení a zrušení. První argument je akce vykonaná při potvrzení a druhý je výchozí obsah políčka.
- Metoda *ShowEnterEmailDialog* slouží k vložení emailové adresy. Je analogický s dialogem pro heslo, jen navíc má i akci po zrušení.

Jako základ pro tvorbu aktivit byla vytvořena abstraktní třída *ActivityNFC*, která je potomkem třídy *ActivitySmart*. Byly v ní zautomatizovány některé úkony, jako je registrace a odregistrování schopnost obsluhovat NFC události a také ona obsluha. Byla takto zavedena metodika, aby v každé takovéto aktivitě byl vytvořen enumerátor na veškeré akce, které se mohou stát po přiložení telefonu a tyto akce byly implementovány jako proměnné typu *RunnableWithParams*. To protože třída obsahuje pole akcí a metody, pro přidání akce do pole, pro vybrání akce jako následující a pak samovolně při vzniku události takovou akci vykoná. Také zajišťuje vibrování při přikládání telefonu s pomocí třídy *Vibrator*. A navíc obsahuje globální proměnné pro použití dialogů apod.

Bylo rozhodnuto, že pro úkony spojené s nastavováním měření se budou dít moderní formou průvodce, kdy na každé jedné obrazovce se budou volit informace stejného druhu a poté uživatel přejde na další obrazovku a tak dále až do konce. Dalo by se to řešit přepínáním aktivit, ale to je řešení velmi neelegantní, drahé na strojový čas, zbytečně robustní a bylo by možná i problematické z pohledu synchronizace. Android nabízí dobré řešení, a to je použití fragmentů (Fragment), což je vlastně shluk grafických prvků tvořících podmnožinu aktivity. Uvnitř nějakého prvku aktivity lze libovolně přepínat fragmenty.

Byla vytvořena třída *MeasurementActivityActions* děcící třídu *ActivityNFC*, a slouží jako vstupní bod pro všechny nabídky realizované průvodcem. Aktivita se totiž v průběhu nepřepíná, a proto tato jedna může řešit všechny případy. Pro řešení synchronizace mezi fragmentem

a aktivitou vznikla abstraktní třída *StackFragment* dědící třídu *Fragment*, se smyslem zajistit jakýsi předpis pro fragment, který bude mít synchronizační body (metody) a půjde dobře slučovat do množin. Třída *MeasurementActivityActions* má enumerátor pro větve, kterými průvodce může směřovat. Má 2 tlačítka, pro pohyb dopředu i zpět a podle stisknutí provádí akce. Na začátku se množina fragmentů naskládá do pole v logickém pořadí a podle tlačítek se přepíná fragment na aktuálním indexu. Tato aktivita si sama řeší tlačítka a tlačítko zpět skrývá, když už není kam se vracet. Do dokončení větve sama skončí. Fragmenty mají právě zmíněné vstupní body proto, aby se aktivita před přepnutím zeptala, zdali vstupní údaje ve fragmentu jsou validní a on souhlasí se svým ukončením, a nebo naopak uživatele v sobě pozdrží. Při stisknutí tlačítka zpět varuje uživatele před ztrátou dat. A po skončení větve může volitelně provést nějakou dokončovací operaci. Tato aktivita taky řeší veškeré NFC operace, protože ty se provádějí až na konci průvodce.

A konečně byla pokryta funkční stránka přístupu k měřicímu systému, aby bylo možno řídit měření. Pro reprezentaci měření, jakožto úlohy s určitými parametry, byl zaveden vlastní objekt, jehož členy jsou informace a jehož metody slouží k dopočítání chybějících informací. Definuje jej třída *MeasurementObject*, jejíž členy jsou zobrazeny ve výpisu 22.

```
public MeasurementStates MeasurementState;
public MeasurementTypes MeasurementType;
public Date MeasurementStart;
public Date MeasurementEnd;
public int Interval;
public int SampleCount;
public int BytesPerSample;
public int MaxSampleCount;
public double Coefficient;

public enum MeasurementStates
{
   None,
   MCUEvaluationPending,
   WaitingForStartTime,
   Running,
   FinishedSuccessfullyAuto,
   FinishedSuccessfullyManual,
   Failed,
   FailedLowBattery
}
```

```
public enum MeasurementTypes
{
   None,
   Voltage
}

public void LearnTypeDependentInfo()
public void LearnSampleCountFromEndDate()
public void LearnEndDateFromSampleCount()
```

Výpis 22: Přehled veřejných členů třídy MeasurementObject

Stojí za zvláštní povšimnutí, že oba enumerátory mají přesně stejné položky ve stejném pořadí, jako enumerátory v jazyce C na MCU. Jak již bylo zmíněno, tento způsob je nutný pro správnou číselnou reprezentaci. Většina proměnných svým názvem mluví za vše. Objasnit je nutné asi tak proměnnou *Coefficient*, která obsahuje konstantu na přepočet hodnoty vzorku do fyzikálního rozsahu senzoru. Všechny 3 metody slouží k tomu, že si třída dopočítá chybějící údaje z již dodaných údajů. Metoda *LearnTypeDependentInfo* podle měřené veličiny přiřadí počet bajtů na vzorek, přepočtový koeficient a spočítá maximální počet vzorků, které paměť obsáhne. Ostatní 2 metody jsou k sobě duální a slouží z zjištění, jak dlouho bude měření trvat, jedna po stránce počtu vzorků a druhá po stránce kalendářní.

Hlavní a velmi robustní ryze statická třída této sekce je *MeasurementSystem*, která má na starost zabezpečení a autorizaci; zjišťování informací o měření, jeho zahájení i zastavení; nasbírání naměřených dat a jejich úpravu; mazání paměti, generování, ukládání, zasílání i mazání CSV souborů; a také formátování časových údajů. Přehled veřejných členů s označením sekcí je ve výpisu 23.

```
public static void Init(SharedPreferencesManager settings)

//Protection
public enum ProtectionType
{
   NoProtection,
   WriteProtection,
   WriteReadProtection
}
public static ProtectionType CurrentProtection;
public static boolean WriteAuthorized;
public static void GetProtectionInfo(OperationResult result, Intent intent)
```

```
public static void Authorize(M24SR.AccessType type, OperationResult result,
    Intent intent)

//Measurement
public static int BatteryPercentage;
public static MeasurementObject CurrentMeasurement;
public static int TakenSamplesAmount;
public static int MemoryBytesUsed;
public static long[] MeasuredValues;
public static double[] RecalculatedValues;
public static String CSV;
public static String Email;
public static void GetMeasurementInfo(OperationResult result, Intent intent)
public static void SetMeasurementInfo(OperationResult result, Intent intent)
public static void StopMeasurement(OperationResult result, Intent intent)
public static void GetMeasurementData(OperationResult result, Intent intent)
public static void RecalculateValues()
public static void ClearStorage(boolean whole, OperationResult result, Intent
    intent)

//CSV
public static void GenerateCSV()
public static boolean SaveCSV()
public static boolean SendCSV()
public static int DeleteAllCSVs()

//Formatting
public static int DateFormat;
public static String DateToString(Date date)
```

Výpis 23: Přehled vybraných veřejných členů třídy MeasurementSystem

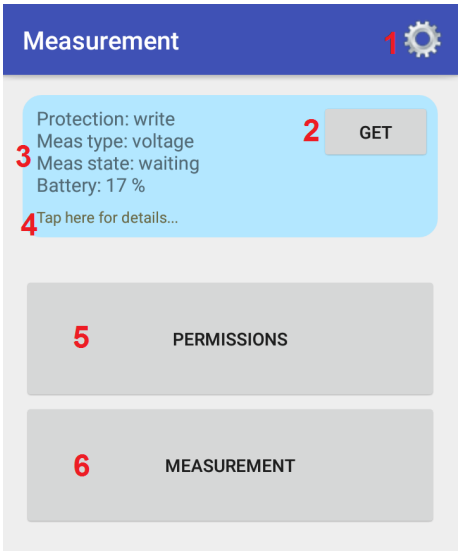
Bylo by příliš rozsáhlé komentovat všechny členy, většina z nich o sobě vypovídá názvem. Co stojí za povšimnutí je fakt, že není použit přístup běžný v Javě, kdy práci s každou proměnnou bývají nějaké vstupní metody. Namísto toho jsou proměnné veřejné a metody, které se k nim vztahují, třeba i postrádají argumenty. Tento způsob byl zvolen kvůli použití průvodců a přístupů z různých míst. Aby se nemusely stále předávat reference, statická třída v sobě drží všechny hodnoty a ty jsou nastavovány v z libovolných míst (např. z jednotlivých obrazovek průvodce)

a nakonec jsou metodami nějak uplatněny. Všechny metody jen nějakým způsobem aplikují doposud popsaný aparát.

6.2.4 Výsledná aplikace

Tato závěrečná sekce vývoje aplikace je už jen o tvorbě grafického rozhraní v rozumně rozřazených nabídkách a ladění pro pohodlnost. Náplň této části implementace spočívala v opakování standardních kroků, jako je tvorba aktivit a fragmentů, a s tím souvisejících layoutů. Jedná se dosti o design, než o intelektuální vývoj. Proto v této části budou popisovány již sekce hotové aplikace s minimálním komentářem.

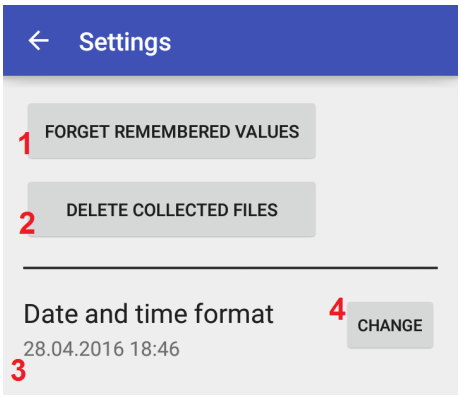
6.2.4.1 Hlavní menu Je vstupním bodem aplikace. Snímek obrazovky se nachází na obrázku 11. Ikona 1 vede do menu nastavení. Stisknutí tlačítka 2 je základním úkonem, neboť po následném zobrazení výzvy a přiložení telefonu dojde k načtení informací z paměti čipu a políčka se zobrazí. Tento snímek je už po načtení, jinak by dolní tlačítka (5, 6) byla zablokovaná. Při kliknutí na text 4 se otevře dialog s podrobnějšími informacemi o stavu měření. Tlačítko 5 vede do nabídky zabezpečení čipu a tlačítko 6 do správy měření.



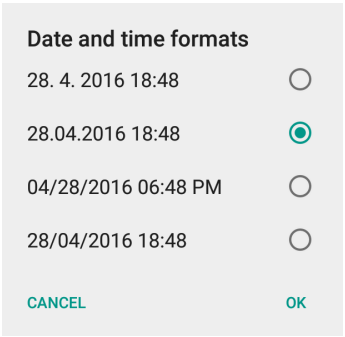
Obr. 11: Hlavní menu

6.2.4.2 Nastavení Umožňuje uživateli spravovat některé vlastnosti. Snímek obrazovky se nachází na obrázku 12. Tlačítko 1 slouží ke smazání zapamatovaných hodnot (SharedPreferences), kterými jsou heslo a emailová adresa. Nejprve se zobrazí dialog s potvrzením. Tlačítko 2 slouží ke smazání všech uložených CSV souborů, nejprve je požadováno potvrzení. Sekce pod

oddělovací čarou slouží k výběru formátu časových údajů zobrazovaných v aplikaci. Na místě 3 je vyobrazen aktuální čas v současném formátu. Ke změně slouží tlačítko 4, které zobrazí nabídku, která je na obrázku 13. V ní se volí formát data a času. Horní možnost je založená na regionálním nastavení, další jsou formáty středoevropský, americký a britský.

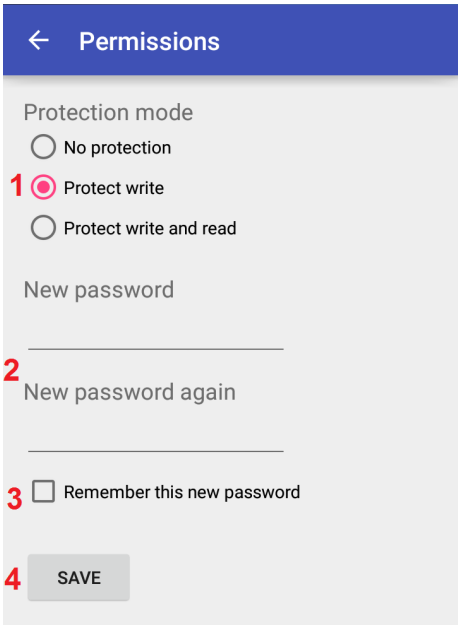


Obr. 12: Nastavení



Obr. 13: Formáty data a času

6.2.4.3 Zabezpečení Vzhled nabídky je zobrazen na obrázku 14. Volí se zde způsob, jakým je paměť čipu chráněna přepínači 1. Lze vybrat mezi nechráněným stavem, ochranou zápisu a sdruženou ochranou zápisu i čtení. Aktivita rozlišuje, z jakého na jaké je zabezpečení měněno a podle toho upravuje zobrazené nabídky. Např. při změně z ochrany zápisu na sdruženou zobrazí přepínač, který volitelně umožňuje změnit heslo (při přepnutí zobrazí potřebná pole). V místech 2 se vyplňuje heslo a pokud je to vhodné, tak se požaduje opakované zadání. Checkbox 3 umožňuje nechat si heslo zapamatovat, aby nemuselo být později zadáváno. Při potvrzení tlačítkem 4 se provádí kontrola hesla a poté je zobrazena výzva k přiložení telefonu.



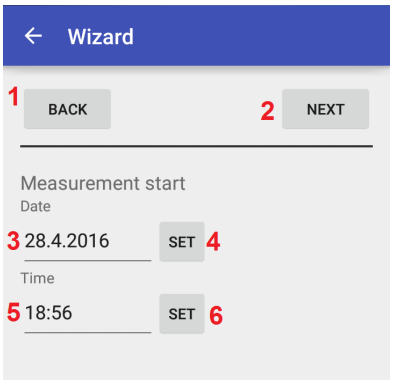
Obr. 14: Nabídka zabezpečení

6.2.4.4 Menu měření Vzhled nabídky je zobrazen na obrázku 15. Jedná se o rozcestník pro další konkrétní nabídky. Z funkčního pohledu se zde kontroluje pouze aktuální zabezpečení při výběru volby a v závislosti na tom, jak je paměť zabezpečena a jaké operace požaduje aktuálně vybíraná volba, se může objevit dialog pro zadání hesla, který bude požadovat přiložení telefonu, aby bylo umožněno pokračovat dále. V závislosti na aktuálním stavu měření mohou být některá tlačítka deaktivovaná a jiná ne.

Obr. 15: Menu měření

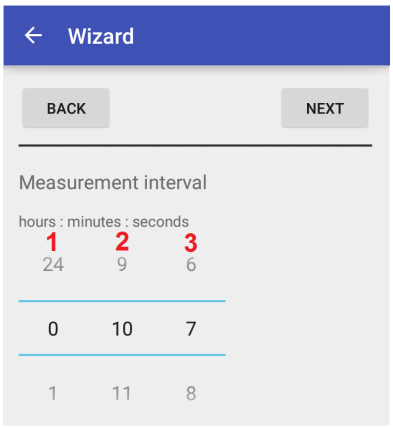
6.2.4.5 Nové měření Započne průvodce se 4 kroky. V prvním kroku se vybírá druh měření (veličina), ten není vyobrazen pro úsporu místa, jedná se pouze o přepínače typu radio.

V druhém kroku se volí, kdy měření započne, tato nabídka je zobrazena na obrázku 16. Je zde vidět, jak funguje univerzální aktivita průvodce, tlačítka 1 a 2 jsou spravována ní samotnou podle situace. Tlačítko 1 slouží k návratu zpět a tlačítko 2 pro přechod na další krok. Toto již nebude znovu popisováno. Tlačítkem 4 se zobrazí kalendář pro výběr data a po výběru se toto datum vloží do pole 3. Tlačítkem 6 se zobrazí hodiny pro výběr času a po výběru se tento čas vloží do pole 5.



Obr. 16: Nové měření 2. krok

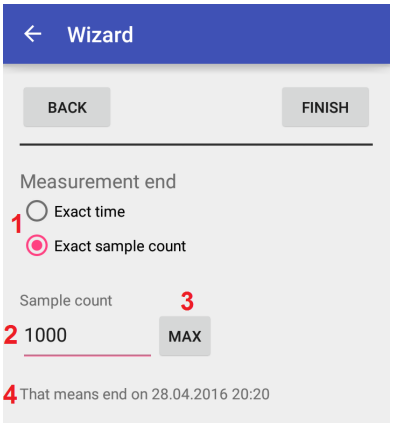
V třetím kroku se vybírá perioda, tato nabídka je vyobrazena na obrázku 17. Perioda znamená interval, po kterém budou prováděna jednotlivá měření, v podstatě se dá hovořit o vzorkovací periodě. Výběr probíhá pomocí 3 číselníků, kdy číselník 1 slouží pro hodiny, číselník 2 pro minuty a číselník 3 pro sekundy. Minimální perioda je 5 s.



Obr. 17: Nové měření 3. krok

Ve čtvrtém kroku se vybírá, jakým způsobem skončí měření a vyobrazen je na obrázku 18. Výběr způsobu se provede tlačítky radio 1 a podle toho se dole mění vstupní pole. V případě výběru konce podle času se čas volí úplně stejně, jako při výběru začátku. Text 4 pak zobrazuje přepočítaný počet vzorků. Naopak při volbě konce podle počtu vzorků slouží pole 2 pro zadání

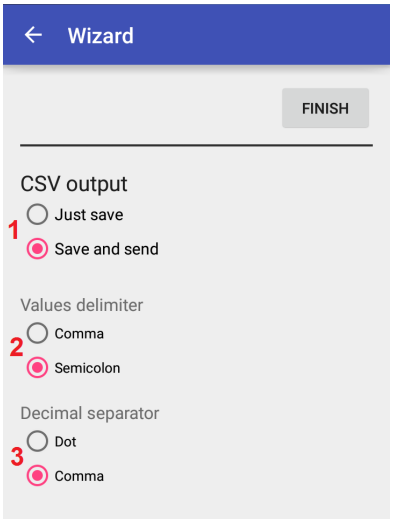
onoho počtu a tlačítko 3 při stisknutí do pole vyplní číslo maximálního možného počtu vzorků. Text 4 pak zobrazuje přepočítané datum konce.



Obr. 18: Nové měření 4. krok

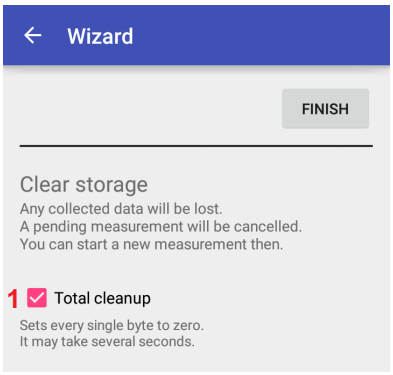
6.2.4.6 Zastavení měření Měření lze takto zastavit před jeho samovolným koncem. Telefon zapíše do paměti příslušný stav měření a uživatel musí počkat, než MCU toto vyhodnotí. Volba zde nebude vyobrazena, neboť průvodce má uměle pouze 1 krok a uvnitř pracovní plochy je pouze napsány, co zastavení znamená, nejsou tam žádné prvky pro uživatelský vstup.

6.2.4.7 Převzetí naměřených dat Probíhá v nabídce průvodce, který má 2 kroky, z nichž teprve druhý je akční. Ten je také vyobrazen na obrázku 19. Zde dojde k výstupu dat z aplikace. Přepínači 1 se vybírá, zdali se CSV soubor pouze uloží, nebo bude i zaslán na emailovou adresu. V případě druhé volby vyskočí dialog pro zadání. Přepínače 2 pak vybírají, zdali oddělovačem v souboru bude čárka či středník a přepínače 3 vybírají, jestli se desetinná čísla budou rozdělovat tečkou či čárkou. Tyto volby umožňují vyhovět tabulkovému procesoru Microsoft Excel v lokalizované verzi. Při zaslání na email je zapotřebí mít připojení k internetu. To se kontroluje příkazem ping na Google DNS server a v případě neúspěchu je uživatel vyzván k nápravě. V případě úspěchu je uživateli brzy doručen email s odkazem na PHP skript, který soubor předá ze svého úložiště.



Obr. 19: Převzetí naměřených dat

6.2.4.8 Vyčištění paměti Průvodce má v tomto případě jediný krok, vyobrazený na obrázku 20. Jsou zde zobrazeny informace pro uživatele a zaškrtávací políčko 1 mění způsob smazání. V základě se pouze nastaví čipu velikost obsazené paměti na nulovou. V případě, že je políčko zaškrtnuto, tak se nejprve paměť nastaví na zaplněnou a každý jednotlivý bajt je změněn na nulu, pak se teprve paměť zmenší na prázdnou. Tato volba se dá označit za bezpečnostní.



Obr. 20: Vyčištění paměti

6.3 Jednoúčelová aplikace pro Windows

Jak již bylo vysvětleno v návrhu, aplikace pro Android není schopná nastavit mikrokontroléru čas s dostatečnou rychlostí, a tento problém je dobře odstranitelný velmi malou aplikaci pro Windows, jakožto nejrozšířenější OS pro PC. Nejsnazší a nejrychlejší vývoj nabízí platforma

.NET, a proto byla zvolena. Cílem je zde, aby aplikace odeslala po sběrnici UART (sériová linka) časový řetězec. Výchozím předpokladem je tedy implementace sběrnice.

6.3.1 Funkční základ

U sběrnic je základním předpokladem synchronizace existence bufferu. Pro tento účel byla vytvořena generická třída *FIFO*, jejíž veřejné členy jsou ve výpisu 24. Není zapotřebí zvláštního komentáře, buffer je známá konstrukce vyskytující se v mnoha různých prostředích.

```
public FIFO()
public void Enqueue(T item)
public bool Dequeue(ref T output)
public void Clear()
```

Výpis 24: Veřejné členy třídy FIFO

S využitím této třídy mohla vzniknout třída *UART*, která pokrývá plnou funkčnost sběrnice. Využívá zabudované komponenty *SerialPort*, která by sama o sobě mohla stačit, ale tato vlastní třída udělá použití snazším. Její veřejné členy jsou zobrazeny ve výpisu 25. Značné usnadnění přináší její metoda *FillPortNamesCombo*, která do dodaného prvku vyjížděcí nabídky doplní sériové porty registrované operačním systémem. Je vhodné volat ji z události rozjíždění. Některé metody naopak potřebné nebudou, protože pro tuto aplikaci není zapotřebí ze sběrnice přijímat data, stačí je pouze odesílat.

```
public UART()
static public string[] ListPortNames()
static public int[] ListBaudRates()
public void SetPortName(string portName)
public void SetBaud(int baud)
public void SetSendingTimeout(int ms)
public bool IsConnected()
public void Connect()
public void Disconnect()
public void SendLine(string line)
public bool ReceiveLine(ref string output)
public void FlushBuffers()
public static void FillPortNamesCombo(ComboBox combo)
```

Výpis 25: Veřejné členy třídy UART

Použití třídy má několik přirozených zásah. Před otevřením portu je nutné příslušnými metodami s odpovídajícími jmény nastavit vybraný port, vybraný baud rate (pro tuto aplikaci nutně 38400) enumerační položkou a je vhodné nastavit timeout pro odesílání. Poté metoda *Connect* otevře sériový port. Od té chvíle se data posílají metodou *SendLine*, která za textový řetězec přidá řádkový zlom.

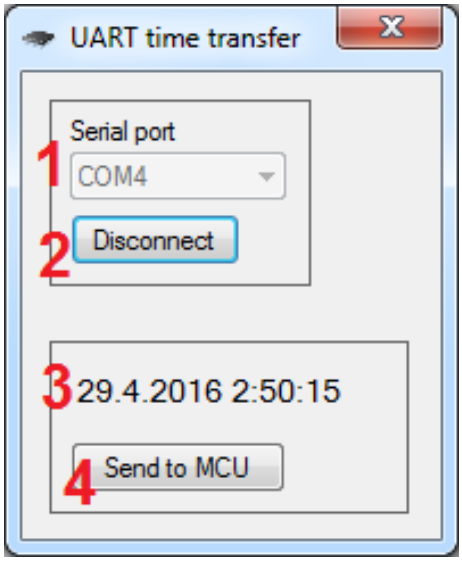
6.4 Výsledná aplikace

Aplikace je dosti jednoduchá a samotný layout chudý. Obsahuje rozjížděcí nabídku pro výběr portu, tlačítko pro připojení či odpojení, text s aktuálním datem a časem, a nakonec tlačítko pro odeslání časové informace po sběrnici. O aktualizaci času v textovém poli se stará časovač (komponenta Timer) a o odeslání času také.

Po stisknutí tlačítka odeslání se od systému vezme aktuální datum a čas, a tento údaj se převede do číselné reprezentace (.NET používá tzv. souborový čas, což je počet stovek nanosekund, který uběhly od 1.1.1601). Zjistí se nejbližší čas s celistvým počtem sekund, a rozdílem s aktuálním časem se získá počet milisekund, které zbývají do celistvého počtu sekund. Toto je časovači vloženo jako interval a je spuštěn odpočet.

V obslužné metodě události časovače se naformátuje časový řetězec do výstupního formátu `set_time ddMMyyHHmmss` a odešle se po sériové lince. Následně se pošle řetězec `identify`. První řetězec způsobí, že KL25Z si tuto informaci rozkouskuje a uloží si ji do hodin reálného času. Druhý řetězec vyvolá na KL25Z blikání RGB LED diody po dobu několika sekund, takže uživatel vidí, zdali úspěšně vložil čas.

Výsledná podoba aplikace je na obrázku 21. Rozjížděcí nabídka při otevření zobrazí všechny dostupné porty a uživatel vybere ten správný, načež stiskne tlačítko 2. Tím se výběr portů zablokuje a povolí se tlačítko 4. V textu 3 uživatel neustále vidí aktuální čas. Po stisku tlačítka 4 se zablokuje celé rozhraní, dokud nenastane celá sekunda a nepošle se údaj. Pokud k tomu úspěšně došlo, uživatel vidí LED diodu na desce KL25Z blikat.



Obr. 21: Aplikace pro zaslání času

7 Testování řešení

Celé řešení sestává ze 3 menších celků, které kvůli své provázanosti musí být testovány společně, avšak poté je možné zaměřit se na vlastnosti konkrétního z nich.

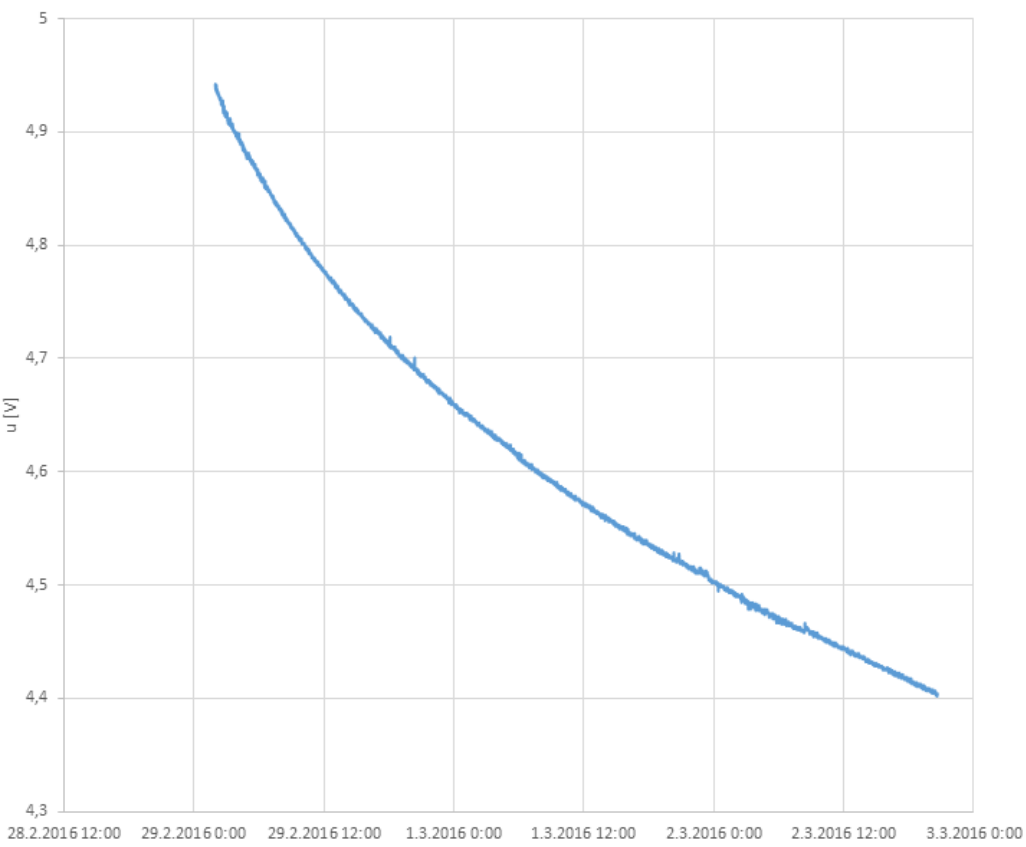
Nejjednodušší pro podrobení testu je aplikace pro Windows. Ta má pro svou funkčnost jednu prerekvitizu, a tou je přítomnost runtime knihoven frameworku .NET, což zpravidla nebývá problém, neboť ho požaduje mnoho dalších aplikací, a tak bývá v systému nainstalován. Aplikace byla otestována na desktopovém počítači s výkonným HW i na notebooku Lenovo nižší kategorie, na obou pod operačním systémem Windows 7. V obou případech fungovala bezchybně. Zjistila všechny dostupné porty a při výběru správného neměla problém se připojit. Při výběru obsazeného či jiné chybě ohlásila chybu, bez ohrožení stability. Odeslání časové informace bylo také bezchybné. Vzhledem k jednoduchosti aplikaci nelze podrobit komplexnějším testům.

Aplikace pro systém Android byla jakožto softwarový produkt podrobena testu metodou černé a bílé skříňky, a to na telefonu Huawei P8 Lite s OS Android Lollipop 5.0.1, na kterém byl prováděn vývoj aplikace. Při metodě černé skříňky byla aplikace používána běžným způsobem a byla opakovaně vytvářena měření, zastavována, vyzvedávána a mazána data, měněno nastavení apod. tak, aby byl simulován běžný uživatel. Při tomto testu byla funkčnost zhodnocena jako bezchybná. Poté byl proveden test metodou bílé skříňky, kdyby byly cíleně procházeny všechny možnost, byly nastavovány všechny kombinace voleb a úmyslně byly zadávány nekorektní či neideální vstupy. Výsledek byl pozitivní, funkčnost byla bezchybná, nepodařilo se vyvolat žádný problém.

Poslední částí k otestování je mikrokontrolér. První pohled pro test je bezprostřední funkčnost při vytvoření náročných podmínek. Bylo otestováno chování při rychlých pohybech telefonu poblíž antény i setrvání ve chvíli, kdy je zapotřebí zapisovat do paměti data. Ukázala se správná ošetřenost, neboť nevznikl problém. Opakovaně bylo testováno správné časování měření, nenastala situace, aby nedošlo ke změření. Byly úmyslně použity akce aplikace, které předpokládaly data, které již byly neaktuální (např. status na telefonu byl před započetím měření, avšak to už započalo), také byla ověřena stabilita. Nejhorší problém, který v tom případě nastane, je ztráta dat, ne však žádné nedefinované chování, které by vedlo k selhání.

Druhý pohled je ten, že na rozdíl od obou aplikací je u programu na KL25Z přítomen jev, který lze označit jako stárnutí instance. Program totiž běží nepřetržitě (samozřejmě hlavní smyčka neběží při úsporném režimu, avšak stále běží komponenty), a mění se hodnoty proměnných, mění se čas a může kvůli nedokonalostem docházet k postupnému nevhodnému vývoji stavů a tím k selháním. Především v případě, že měření je načasováno se značným odkladem a přitom je dlouhodobého charakteru, je ohroženo těmito problémy. Rizikovou částí je práce s datem a časem, protože dochází k přelomům časových celků. Pro vypovídající test se zohledněním těchto faktorů, který otestuje celkovou funkčnost a účelnost, bylo provedeno dlouhodobé moni-

torování napětí na kondenzátoru. Byl vybrán elektrolytický kondenzátor s kapacitou 1000 µF, který byl nabyt na napětí 5 V a bylo načasováno jeho měření, které započalo 29.2.2016 ve 2:00 a skončilo 2.3.2016 ve 20:39. Vzorkovací perioda byla 1 min a nasbíráno bylo 4000 vzorků, čímž byla zaplněna celá paměť. Bylo tedy monitorováno samovolné vybíjení. Po dokončení byl aplikací vygenerován CSV soubor, ze kterého byl tabulkovém procesoru Microsoft Excel vytvořen graf, který je zobrazen na obrázku 22. Časy odpovídaly nastaveným hodnotám a naměřená data vypadají věrohodně. Především nedošlo k selhání v průběhu. Což je o to lepší z důvodu, že datum je poměrné výjimečného charakteru, protože byl zrovna konec února v přestupném roce, což je velký benchmark pro algoritmy na posouvání data, a prokazuje jejich bezchybnost. Tím se dá systém prohlásit za funkční.



Obr. 22: Graf monitorovaného napětí kondenzátoru

8 Závěr a zhodnocení

Dovolte mi rekapitulovat cíle této práce. V teoretické části jsem se měl zabývat rozborem zadané problematiky, což jsem provedl, snad s udržením věcnosti a sám jsem se poučil a získal nové znalosti, což považuji za přínosné.

Nejzákladnějším cílem praktické části bylo implementovat vhodným a přínosným způsobem RFID/NFC technologii pro nízkoodběrové embedded zařízení. Protože snahou bylo využít technologie užitečně, a ne jen samoúčelně, upřesněným cílem bylo vytvoření měřicího systému pro dlouhodobá měření.

Jsem skálopevně přesvědčen, že těchto cílů jsem dosáhl elegantním řešením. Pro mikrokontrolér KL25Z jsem vytvořil program, který zajišťuje samočinné (není myšleno ve smyslu autonomní) měření s použitím takové metodiky, že byly zavedeny nízkoodběrové algoritmy. Pro OS Android jsem vytvořil funkční a uživatelsky přívětivou aplikaci pro ovládání tohoto měřicího programu. A pro OS Windows jsem zhotovil aplikaci, která řeší předání času měřicímu programu. Dohromady tyto části tvoří funkční měřicí systém, který byl náležitě otestován a vykázal bezproblémovou činnost.

Použití pasivního NFC tagu v takovémto systému je moderním přístupem k této technologii a správným směrem jejího využití. Především se ukazuje její přínos pro nízkoodběrovost, neboť je energetická závislost jistým způsobem přesměrována na vnější zdroj. Taktéž použití chytrého telefonu pro řízení jakéhokoliv systému či zařízení je přístup velice dnešní a pro koncového uživatele značně atraktivní, což také považuji za výhodu.

Troufnu si tedy finálně prohlásit práci za úspěšnou a vyjadřuji nad tímto faktem potěšení.

Literatura

[1] YIU, Joseph. *The definitive guide to the ARM Cortex-M0.* Burlington, MA: Newnes, 2011, xxii, 529 p. ISBN 0123854776.

[2] OSHANA, Robert a Mark KRAELING *Software engineering for embedded systems: methods, practical techniques, and applications.* Maltham, MA: Newnes, 2013. ISBN 0124159176.

[3] PRAUZEK, M., P. MUSILEK, A. G. WATTS a M. MICHALIKOVA. Powering Environmental Monitoring Systems in Arctic Regions: A Simulation Study. *Elektronika ir Elektrotechnika* [online]. 2014, 20(7), - [cit. 2016-01-19]. DOI: 10.5755/j01.eee.20.7.8020. ISSN 2029-5731. Dostupné z: http://eejournal.ktu.lt/index.php/elt/article/view/8020

[4] WATTS, Asher G., Michal PRAUZEK, Petr MUSILEK, Emil PELIKAN a Arturo SANCHEZ-AZOFEIFA. Fuzzy power management for environmental monitoring systems in tropical regions. In: 2014 *International Joint Conference on Neural Networks (IJCNN)* [online]. IEEE, 2014, s. 1719-1726 [cit. 2016-01-20]. DOI: 10.1109/IJCNN.2014.6889844. ISBN 978-1-4799-1484-5. Dostupné z: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6889844

[5] BRANDŠTETTER, Pavel. *Elektronika.* Ostrava: Vysoká škola báňská - Technická univerzita, 2007, 1 CD-R. ISBN 978-80-248-1481-0.

[6] GLOVER, Bill a Bhatt HIMANSHU. *RFID essentials.* 1st ed. Beijing: O'Reilly, 2006, xiii, 260 s. ISBN 0-596-00944-5.

[7] STYGER, Erich. Tutorial: Using the FRDM-KL25Z as Low Power Board. In: *MCU on Eclipse* [online]. Horw, 2013 [cit. 2016-01-20]. Dostupné z: http://mcuoneclipse.com/2013/10/20/tutorial-using-the-frdm-kl25z-as-low-power-board/

[8] KEATING, Michael. *Low power methodology manual: for system-on-chip design.* New York, NY: Springer, 2007, xvi, 300 p. ISBN 9780128016305.

A Projekt pro KL25Z

Projekt vytvořený v prostředí CodeWarrior je přiložen na CD v sekci příloh ve složce „CodeWarrior“.

B Projekt pro Android

Projekt vytvořený v prostředí Android Studio je přiložen na CD v sekci příloh ve složce „AndroidStudio“.

C Projekt pro Windows

Projekt vytvořený v prostředí Microsoft Visual Studio 2013 je přiložen na CD v sekci příloh ve složce „VisualStudio“.